UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM *FRAMEWORK* DE TESTES UNITÁRIOS PARA PROCEDIMENTOS DE CARGA EM AMBIENTES DE *BUSINESS INTELLIGENCE*

IGOR PETERSON OLIVEIRA SANTOS

SÃO CRISTÓVÃO/SE
2016

UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

IGOR PETERSON OLIVEIRA SANTOS

UM *FRAMEWORK* DE TESTES UNITÁRIOS PARA PROCEDIMENTOS DE CARGA EM AMBIENTES DE *BUSINESS INTELLIGENCE*

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PROCC) da Universidade Federal do Sergipe (UFS) como parte de requisito para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Methanias Colaço Rodrigues Júnior

SÃO CRISTÓVÃO/SE
2016

IGOR PETERSON OLIVEIRA SANTOS

UM *FRAMEWORK* DE TESTES UNITÁRIOS PARA PROCEDIMENTOS DE CARGA EM AMBIENTES DE *BUSINESS INTELLIGENCE*

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação (PROCC) da Universidade Federal do Sergipe (UFS) como parte de requisito para obtenção do título de Mestre em Ciência da Computação.

BANCA EXAMINADORA

Prof. Dr. Methanias Colaço Rodrigues Júnior, Presidente
Universidade Federal de Sergipe (UFS)

Prof. Dr. Alberto Costa Neto
Universidade Federal de Sergipe (UFS)

Prof. Dr. Paulo Caetano da Silva
Universidade Salvador (UNIFACS)

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL UNIVERSIDADE FEDERAL DE SERGIPE

Santos, Igor Peterson Oliveira
S237f Um framework de testes unitários para procedimentos de carga em ambientes de business intelligence / Igor Peterson Oliveira Santos; orientador Methanias Colaço Rodrigues Júnior. – São Cristóvão, 2016.
102 f. : il.

Dissertação (mestrado em Ciências da computação) – Universidade Federal de Sergipe, 2016.

1. Programas de computador - Testes. 2. Framework (Arquivo de computador). 3. Engenharia de software. 4. Software - Testes. 5. Inteligência competitiva (Administração). I. Rodrigues Júnior, Methanias Colaço. II. Título.

CDU 004.4

UM *FRAMEWORK* DE TESTES UNITÁRIOS PARA PROCEDIMENTOS DE CARGA EM AMBIENTES DE *BUSINESS INTELLIGENCE*

Este exemplar corresponde à redação da Dissertação de Mestrado, sendo o Exame de Defesa do mestrando **IGOR PETERSON OLIVEIRA SANTOS** aprovado pela Banca Examinadora.

São Cristóvão - SE, 30 de AGOSTO de 2016

Prof. Dr. Methanias Colaço Rodrigues Júnior
Orientador

Prof. Dr. Alberto Costa Neto
Membro

Prof. Dr. Paulo Caetano da Silva
Membro

RESUMO

A qualidade de um produto de *software* está diretamente relacionada com os testes empregados durante o seu desenvolvimento. Embora os processos de testes para *softwares* aplicativos e sistemas transacionais já apresentem um alto grau de maturidade, estes devem ser investigados para os processos de testes em um ambiente de *Business Intelligence* (BI) e *Data Analytics*. As diferenças deste ambiente em relação aos demais tipos de sistemas fazem com que os processos e ferramentas de testes existentes precisem ser ajustados a uma nova realidade. Neste contexto, grande parte das aplicações de *Business Intelligence* (BI) efetivas depende de um *Data Warehouse* (DW), um repositório histórico de dados projetado para dar suporte a processos de tomada de decisão. São as cargas de dados para o DW que merecem atenção especial relativa aos testes, por englobar procedimentos críticos em relação à qualidade. Este trabalho propõe uma abordagem de testes, baseada em um *framework* de testes unitários, para procedimentos de carga em um ambiente de BI e *Data Analytics*. O *framework* proposto, com base em metadados sobre as rotinas de carga, realiza a execução automática de casos de testes, por meio da geração de estados iniciais e a análise dos estados finais, bem como seleciona os casos de testes a serem aplicados. O objetivo é melhorar a qualidade dos procedimentos de carga de dados e reduzir o tempo empregado no processo de testes. A avaliação experimental foi realizada através de dois experimentos controlados executados na indústria. O primeiro avaliou a utilização de casos de testes para as rotinas de carga, comparando a efetividade do *framework* com uma abordagem manual. O segundo experimento efetuou uma comparação com um *framework* genérico e similar do mercado. Os resultados indicaram que o *framework* pode contribuir para o aumento da produtividade e redução dos erros de codificação durante a fase de testes em ambientes de suporte à decisão.

Palavras-chave: *Business Intelligence*; *Data Warehouse*; Testes de *Software*; *Data Analytics*; Engenharia de Software Experimental.

ABSTRACT

Business Intelligence (BI) relies on Data Warehouse (DW), a historical data repository designed to support the decision making process. Despite the potential benefits of a DW, data quality issues prevent users from realizing the benefits of a BI environment and Data Analytics. Problems related to data quality can arise in any stage of the ETL (Extract, Transform and Load) process, especially in the loading phase. This thesis presents an approach to automate the selection and execution of previously identified test cases for loading procedures in BI environments and Data Analytics based on DW. To verify and validate the approach, a unit test *framework* was developed. The overall goal is achieve data quality improvement. The specific aim is reduce test effort and, consequently, promote test activities in DW process. The experimental evaluation was performed by two controlled experiments in the industry. The first one was carried out to investigate the adequacy of the proposed method for DW procedures development. The Second one was carried out to investigate the adequacy of the proposed method against a generic *framework* for DW procedures development. Both results showed that our approach clearly reduces test effort and coding errors during the testing phase in decision support environments.

Keywords: Business Intelligence; Data Warehouse; Software Testing; Data Analytics; Experimental Software Engineering.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE SIGLAS

BI	*Business Intelligence*
ETL	*Extraction, Transformation and Load*
DW	*Data Warehouse*
ES	*Engenharia de Software*
OLTP	*Online Transaction Processing*
OLAP	*On-line Analytical Processing*
FTUnit	*Framework* de Testes de Unidade

SUMÁRIO

CAPÍTULO 1

1. INTRODUÇÃO

1.1 Contextualização

A informação representa fator crucial para as empresas no processo de tomada de decisões. Visto que as mudanças no mundo dos negócios são constantes, as organizações veem que a informação não representa um mero resultado de transações, mas um propulsor para atingir melhores resultados. Para auxiliar as áreas estratégicas das organizações, o ambiente de *Business Intelligence* (BI, ou Inteligência Aplicada aos Negócios) apresenta-se como um conjunto de tecnologias que permitem o cruzamento de informações e suportam a análise dos indicadores de desempenho de um negócio (COLAÇO, 2004).

Uma fonte de dados responsável por fornecer informações, de forma eficiente, prática e com qualidade, é chamado de *Data Warehouse* (DW). Este representa um banco de dados histórico, separado lógica e fisicamente do ambiente de produção de uma organização, concebido para dar suporte às análises e decisões gerenciais. O objetivo dessa abordagem é selecionar, integrar e organizar dados provenientes do ambiente operacional e fontes externas para que possam ser acessados de forma mais eficiente e para que possam representar uma única realidade da organização (KIMBALL, 2008) (INMON, 2005) (COLAÇO, 2004).

Os problemas com qualidade de dados podem surgir em várias fases do processo conhecido como ETL (*Extract, Transform and Load*), especialmente no estágio de carga. As principais causas, durante a fase de carga, que contribuem para a má qualidade dos dados são apontadas em (RANJIT; KAWALJEET SINGH, 2010). Dentre essas causas, podemos destacar: a) Implementações incorretas ou a má interpretação das estratégias de armazenamento de histórico nas dimensões; b) Falta de tratamento em valores nulos; c) Tratamento incorreto de colunas de auditoria; d) Perda de dados (dados rejeitados) durante o processo de carregamento.

O teste de procedimentos ETL é considerado a fase de teste mais crítica e complexa no ambiente de DW, pois ela afeta diretamente a qualidade dos dados (GOLFARELLI; RIZZI, 2009). Os procedimentos ETL, mais precisamente as rotinas voltadas para carga, podem ser vistos como aplicações de banco de dados, pois apresentam como entrada o estado inicial de um banco de dados e produzem como saída um novo estado consistente. Essa característica faz com que as rotinas de carga possam ser vistas sob dois aspectos: a) o de Unidade; b) o de Aplicação.

Sob o aspecto de unidade, as rotinas de carga podem utilizar os testes sobre a abordagem *white-box* (caixa-branca) que são aplicadas tipicamente aos componentes e estruturas do programa. Para o aspecto de aplicação, a abordagem *black-box* (caixa-preta) pode ser a solução para testes nos procedimentos de carga. Nessa abordagem, a preocupação é com a interface da aplicação e não com o comportamento interno e com a estrutura do programa (MYERS; BADGETT; SANDLER, 2012) (PRESSMAN, 2011) (SOMMERVILLE, 2011). Essa abordagem para rotinas ETL, em alguns ambientes, pode ser a única alternativa uma vez que a utilização de ferramentas ETL proprietárias no ambiente de DW produz códigos ou pacotes cuja estrutura interna não é conhecida.

O objetivo principal desta dissertação é propor uma abordagem de testes, baseada em um *framework* de testes unitários, para melhorar a eficiência de carga em um ambiente de *Business Intelligence e Data Analytics*. Utilizando uma abordagem *black-box* e tratando as rotinas de carga sob o ponto de vista de aplicação, o *framework* irá gerar estados iniciais e finais do banco de dados em função de Casos de Testes previamente especificados. O comportamento esperado das rotinas, a seleção dos Casos de Testes e os estados do banco de dados a serem utilizados serão determinados por metadados utilizados pelo *framework*.

Está sendo proposta uma abordagem, ao invés de um método, pois, segundo o (IEEE, 2010), um método "é um padrão que descreve as características do processo ou procedimento ordenado, utilizados na engenharia de um produto ou de um serviço de realização". Neste contexto, nossa abordagem descreve como empresas poderão utilizar metadados para executar casos de testes em ambientes de BI, baseadas em um modelo conceitual que promove a facilitação deste processo. Foi desenvolvida uma ferramenta para automação de parte do processo, mas sem a definição ou imposição de uma ordem, o que nos fez declinar quanto ao termo "método".

Então, para concepção da abordagem, foi realizada uma revisão, com características sistemáticas, sobre as idiossincrasias dos procedimentos de carga de dimensões em um ambiente de DW. A partir do levantamento dessas especificidades, foram definidos os Casos de Testes que serviram de base para a realização de testes automáticos. Em seguida, construiu-se o *framework* de testes de unidades, seguido da automatização da execução dos Casos de Testes num ambiente de BI.

1.2 Análise do problema

Alguns trabalhos já apresentam problemas quanto à falta de disponibilidade de testes de unidade automáticos em ferramentas ETL que também é apontada como causa para a má qualidade dos dados (RANJIT; KAWALJEET SINGH, 2010). Não é difícil perceber que existe uma relação entre a falta de testes unitários e as demais causas já apresentadas. Na verdade, o processo de testes no ambiente de *Data Warehouse* como um todo tem sido negligenciado (GOLFARELLI; RIZZI, 2009).

O trabalho apresentado em (Krawatzeck, 2013 apud Krawatzeck et al, 2015) mostra um *survey* online realizado em 2013 com 870 profissionais de BI em países de língua alemã. O resultado dessa pesquisa aponta que os testes em ambientes de DW estão sendo negligenciados atualmente. Há uma proporção em que desse total entrevistado, apenas 55 responderam que há algum tipo de tratamento de testes nesse ambiente para prover qualidade de dados (taxa de resposta = 6,3%). Desses que responderam usar testes em ambientes de DW, 54% informaram que o testam de qualquer maneira, uma vez que consideram o baixo orçamento e a falta de ferramentas gratuitas no mercado para realizar testes em geral ou automatizados em DWs. Além dos desafios acima mencionados, a negligência dos testes nesse ambiente é particularmente surpreendente, uma vez que pesquisas anteriores expõem que muitos DWs tiveram taxas de falha de mais de 50% (Schutte et al, 2011 apud Krawatzeck et al, 2015).

O problema desse projeto é investigar a possibilidade de maior eficiência e eficácia da execução de testes em procedimentos de carga utilizando um *framework* de testes. Dessa forma, foram definidas as seguintes perguntas para a pesquisa:

a) Questão 1: a execução de testes automáticos pode aumentar a produtividade dos programadores durante o processo de teste em um DW?

b) Questão 2: a execução de testes automáticos pode reduzir ou eliminar erros nas rotinas de carga em um DW?

c) Questão 3: em um contexto de carga para DW, os casos de testes apresentam melhor desempenho utilizando o *framework* de testes comparado com um *framework* genérico?

Logo, temos três hipóteses:

a) Hipótese 1

Pretendemos negar a hipótese H0:

- Hipótese nula H0: a execução de testes automática e manual tem a mesma eficiência.
- Hipótese alternativa H1: a execução de testes automática é mais eficiente que a execução de testes manual.

b) Hipótese 2

Pretendemos negar a hipótese H0:

- Hipótese nula H0: a execução dos testes automática e manual tem a mesma eficácia.
- Hipótese alternativa H1: a execução de testes automática é mais eficaz que a execução de testes manual.

c) Hipótese 3

Pretendemos negar a hipótese H0:

- Hipótese nula H0: a execução dos casos de testes para o *framework* de testes e o *framework* genérico tem a mesma eficácia.
- Hipótese alternativa H1: a execução dos casos de testes para o *framework* de testes é mais eficaz que a execução no *framework* genérico.

1.3 Justificativa

Este projeto busca justificar a adoção de processos de testes no ambiente de BI, especialmente na fase de carga. A pouca utilização de processos de testes num DW é creditada às diferenças entre a arquitetura desse ambiente e as arquiteturas dos sistemas Aplicativos e sistemas Transacionais. Essas diferenças fazem com que as técnicas de testes utilizadas por estes últimos precisem ser ajustadas para um ambiente de DW, conforme apontam vários autores (GOLFARELLI; RIZZI, 2009) (ELGAMAL, 2013) (DESHPANDE, 2013).

Sendo assim, este projeto apresenta uma abordagem *black-box* para testes automáticos em rotinas de carga. Os resultados apontam contribuições para a adoção de estratégias de testes no ambiente de *Data Warehouse* e, consequentemente, melhorar a qualidade das rotinas de carga de dados e a redução do tempo gasto nos processos de testes nesse ambiente.

1.4 Objetivos da pesquisa

O objetivo principal desta dissertação é desenvolver e avaliar uma abordagem de testes, baseada em um *framework* de testes unitários, para melhorar a qualidade de procedimentos de carga em ambientes de *Business Intelligence e Data Analytics*.

1.4.1 Objetivos Específicos

Para possibilitar a realização do objetivo geral, podemos enumerar os seguintes objetivos específicos:

- Descrever os Casos de Testes necessários para as rotinas de carga;
- Projetar e construir o *framework* para testes unitários;
- Construir uma ferramenta de testes a partir da extensão do *framework;*
- Realizar dois experimentos:
    - ✓ No primeiro, verificar a utilização de Casos de Testes unitários para procedimentos de carga, comparando a efetividade do *framework* com uma abordagem manual; e,
    - ✓ No segundo, comparar e analisar o *framework* de testes com pelo menos um *framework* similar existente no mercado.

1.5 Metodologia

A metodologia utilizada para o trabalho em questão consiste, em termos de classificação, em uma pesquisa exploratória (SEVERINO, 2008), prática, estudo de caso, de laboratório e experimental. Exploratória, porque serão explorados/estudados materiais acessíveis ao público em geral, como livros, artigos e ferramentas. Prático, pois será implementado um *framework* para realizar testes de unidade nos procedimentos de cargas de dados num ambiente de *Data Warehouse*.

Também será classificada como de laboratório e experimental, devido à execução de experimentos controlados, nos quais serão selecionados grupos de pessoas para utilização da ferramenta desenvolvida.

Quando falamos em experimento, estamos nos referindo a um tipo de pesquisa científica na qual o pesquisador observa a variação de variáveis dependentes por meio da manipulação e controle de uma ou mais variáveis independentes (KERLINGER; LEE, 1973). As variáveis independentes são aquelas que são manipuladas e as Dependentes são o efeito, consequência e o resultado observado da influência das variáveis independentes. Na engenharia de *software*, então, a experimentação refere-se à combinação de fatos, suposições, hipóteses, especulações e crenças que abundam na construção de *software* (JURISTO; MORENO, 2001), sendo manipulados assim, artefatos pertencentes a esta construção, para obtenção de alguma resposta.

Neste contexto, o processo de pesquisa envolveu, inicialmente, uma revisão sistematizada da literatura com objetivo de abordar os principais conteúdos para a definição e concepção do *framework* de testes, apresentado no capítulo 2. Após esta revisão, foram descritos os Casos de Testes para os procedimentos de carga em um ambiente de BI. Em seguida, foram definidos os metadados a fim capturar a semântica necessária para que o *framework* possa selecionar os Casos de Testes a serem aplicados, assim como automatizar a execução dos mesmos.

Com a finalização do projeto da ferramenta e todos os testes realizados na mesma, passamos para o planejamento e execução dos experimentos. A seguir, são enumeradas as etapas de planejamento do primeiro experimento, que teve como objetivo encontrar evidências que possam acatar ou refutar as hipóteses 1 e 2 apresentadas anteriormente na seção 1.1.

1) *Criação do ambiente de Data Warehouse*: nessa fase foi definido e criado o ambiente de DW com os esquemas dimensionais e área de *Staging*. Estes artefatos serviram como base para todo o experimento.

2) *Definição dos Casos de Testes para as rotinas de carga:* foram definidos os casos de testes utilizados pelos programadores do experimento. Esses casos de testes foram aplicados a uma rotina de carga previamente criada.

3) *Revisão de conceitos básicos sobre rotinas de carga para o grupo de programadores*: foi realizada uma revisão sobre as rotinas de carga para ambientes de DW com os programadores selecionados.

4) *Treinamento no framework de testes:* foi realizado um treinamento com os programadores, a fim de que esses pudessem se familiarizar com a ferramenta.

5) *Solicitação de execução dos Casos de Teste de forma manual:* os programadores testaram, manualmente, com base no caso de uso apresentado, o procedimento para as rotinas de carga. Isto serviu para fazer a comparação com o resultado dos testes apoiados pela ferramenta deste projeto.

6) *Solicitação de execução dos Casos de Testes com o apoio do framework:* nesta etapa os programadores utilizaram a ferramenta para testar o procedimento.

Ao final da execução do experimento, foram analisadas as métricas que serviram de base para avaliar as hipóteses 1 e 2: *a) Número de erros de codificação*– número de erros de codificação encontrados nas fases de criação e manutenção dos procedimentos; *b)Tempo gasto no processo de teste* – tempo utilizado durante o processo de teste de rotinas de carga.

Para o segundo experimento que teve como objetivo encontrar evidências que possam acatar ou refutar a hipótese 3 da seção 1.1. As etapas de planejamento seguiram semelhantes às definidas anteriormente até o treinamento do *framework* de testes. A partir deste, as demais etapas são apresentadas a seguir:

5) *Treinamento no framework genérico de testes:* um segundo treinamento foi realizado com os programadores, para que tenham conhecimento da tecnologia.

6) *Solicitação de execução dos Casos de Testes com o apoio do framework:* nesta etapa os programadores utilizaram a ferramenta para testar o procedimento.

7) *Solicitação de execução dos Casos de Testes com o framework genérico:* nesta etapa os programadores utilizaram o *framework* genérico para testar o procedimento.

Ao final da execução do experimento, foi analisada a métrica que serviu de base para avaliar a hipóteses 3: *a) Tempo gasto no processo de teste* – tempo utilizado durante o processo de teste de rotinas de carga.

1.6 Organização da proposta

O texto desta proposta de dissertação estará organizado em 6 capítulos que fornecerão toda a base conceitual e experimental para o entendimento completo da proposta. Os tópicos a seguir descrevem o conteúdo de cada um destes capítulos.

- O Capítulo 1 apresentou esta Introdução;
- O Capítulo 2 apresenta o referencial teórico referente à Arquitetura de um *Data Warehouse*, a qualidade de *softwares* com a utilização de testes de *software* e o relacionamento com o ambiente de BI.
- No Capítulo 3, é apresentada a abordagem proposta neste trabalho, inicialmente sendo mostrado o projeto de casos de testes e, em seguida, o projeto do *framework.* Por fim, é feita a apresentação do *framework* FTUnit, com o módulo de execução de testes em procedimentos.
- O Capítulo 4 apresenta o Planejamento, Operação e Resultados do Experimento I, realizado na indústria com o *framework* de testes para ambientes de BI. O processo experimental é apresentado em forma de artigo. O mesmo foi aceito e publicado no SEKE 2016 (The 28th International Conference on Software Engineering & Knowledge Engineering).
- O capítulo 5 apresenta uma extensão do capítulo 4 que contém a verificação e análise sobre os erros de codificação encontrados no primeiro experimento.
- No capítulo 6, são descritos o Planejamento, Operação e Resultados do Experimento II, o qual tratará da comparação da eficiência do *framework* de testes com um *framework* genérico.
- E, finalmente, o capítulo 7 apresenta as conclusões da Dissertação.

CAPÍTULO 2

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Ambiente de Data Warehouse e Rotinas de Carga

A arquitetura genérica de um ambiente de DW, **Figura 1**, apresenta três principais componentes: o ambiente OLTP (*On-line Transaction Processing*), a área de *Staging* e o *Data Warehouse*. Esses três componentes fazem parte do caminho percorrido pelos dados da organização desde a sua origem até as estruturas finais utilizadas pelos tomadores de decisão.

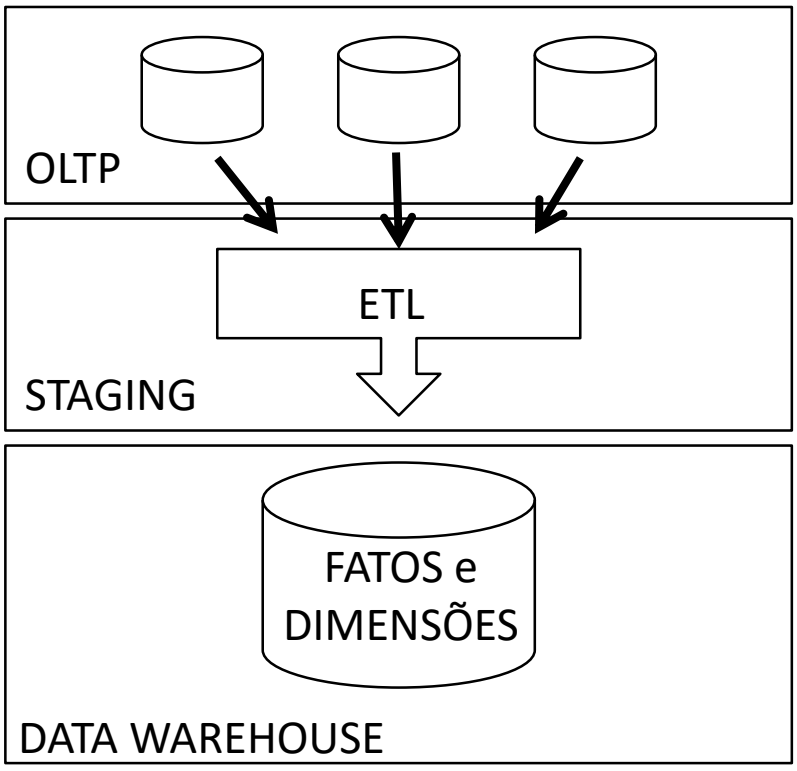

Figura 1: Arquitera genéria de um DW.

O ambiente OLTP é formado pelos sistemas responsáveis pelos processos de negócios diários das organizações. Os dados tratados por esses sistemas também são conhecidos como dados operacionais. Esses dados, que podem ser provenientes de

estruturas internas ou externas à organização, representam as fontes de informações para o DW.

O *Data Warehouse*, que pode ser conceituado como um banco de dados histórico e integrado (COLAÇO, 2004) (KIMBALL, 2002), é a fonte direta de dados utilizados por ferramentas de relatórios, sistemas gerenciais, ferramentas OLAP (*On-line Analytical Processing*) e algoritmos de mineração de dados durante o processo de suporte à decisão. A estrutura interna desse repositório é geralmente formada por esquemas de dados modelados através de uma técnica de projeto lógico conhecida como Modelagem Dimensional (KIMBALL, 2008). Nessa técnica, os esquemas produzidos, conhecidos genericamente com esquemas em estrela, apresentam duas principais estruturas básicas de armazenamento: as tabelas de Fato e as Dimensões.

Antes de serem carregados para os Esquemas em Estrela[1] e disponibilizados para os tomadores de decisões, os dados operacionais precisam passar por um processo conhecido como ETL (*Extract, Transform, Load*), responsável pelas atividades de extração, limpeza, integração e carga. Esse processo ocorre em uma área intermediária conhecida como área de *Staging* (KIMBALL, 2008). Em seguida, a fase final do processo de ETL, a carga de dados, consiste em povoar os esquemas de dados do DW. Em outras palavras, é a fase responsável pela atualização das tabelas de Fatos e das Dimensões.

É na fase de carga que ocorrem os processos mais críticos em relação à qualidade de dados (RANJIT; KAWALJEET SINGH, 2010): a) Geração e gerenciamento de *Surrogate Keys*; b) Manutenção das estratégias de armazenamento de histórico nas Dimensões; c) Carregamento de registros para a tabela de Fatos; d) Carregamento de registros para as Dimensões; e) Manutenção de hierarquias; f) Carregamento e manutenção de Agregados. Explicações detalhadas sobre cada um desses processos podem ser encontradas em (KIMBALL, 2002) (KIMBALL, 2008).

[1] Um Esquema Estrela é a representação dos esquemas de dados com a modelagem dimensional (técnica de projeto lógico de banco de dados) em um DW. O esquema estrela consiste de uma tabela principal, denominada tabela de fatos, e de tabelas auxiliares conhecidas como dimensões (KIMBALL, 2008).

Cada um dos processos mencionados da fase de carga são implementados através de rotinas de carga, construídas manualmente ou com o auxílio de ferramentas. As últimas, geralmente proprietárias, são conhecidas como ferramentas ETL. Quando construídas manualmente, os códigos das rotinas estão disponíveis para serem inspecionados, revisados e testados. As ferramentas ETL, por sua vez, geram, normalmente, programas ou pacotes executáveis. Nesses casos, somente as interfaces de chamadas das rotinas estão disponíveis.

2.2 Teste de *Software*

O teste tem um papel fundamental na diminuição dos custos de manutenção e correção do produto, pois os custos para a correção de um problema aumentam exponencialmente no decorrer do projeto. Quanto mais tarde for detectado um erro, menor será a chance de existir uma solução viável, que cause pouco impacto, que utilize poucos recursos e se adapte ao tempo disponível (PEZZÈ; YOUNG, 2008) (SOMMERVILLE, 2011).

Teste de *software* também contribui bastante na diminuição dos custos diretos, como manutenção, suporte e retrabalho, e também nos custos indiretos, como na qualidade dos produtos e na satisfação dos clientes. E por mais que sejam realizados testes sempre existe a possibilidade do aparecimento de uma falha no produto. No entanto, quanto mais for investido em teste, menor será o custo utilizado na manutenção (PEZZÈ; YOUNG, 2008).

A construção de *software* de alta qualidade requer a combinação de atividades de projeto e verificação ao longo do desenvolvimento. Essas atividades de verificação e projeto podem tomar várias formas, desde aquelas adequadas para a construção altamente repetitiva de itens não críticos para grandes mercados até aquelas para produtos altamente customizados ou altamente críticos (PEZZÈ; YOUNG, 2008).

O *software* é um dos mais complexos e variáveis artefatos construídos de forma regular. Requisitos de qualidade de *software* usados em um ambiente podem ser muito diferentes e incompatíveis para outro ambiente ou domínio de aplicação. A variedade de

problemas e a riqueza de abordagens fazem com que seja um desafio escolher e planejar a combinação correta de técnicas para atingir o nível exigido de qualidade e os requisitos de custo.

2.2.1 Atividade de Teste de *Software*

A atividade de teste de *software* é um elemento crítico e representa a última revisão de especificação, projeto e codificação. O *software* tendo um destaque crescente como elemento de sistema e os "custos" envolvidos associados às falhas encontradas no *software*, são forças propulsoras para uma atividade de teste bem cuidadosa e bem planejada (PEZZÈ; YOUNG, 2008).

Para o engenheiro de *software* a atividade de teste constitui uma anomalia interessante. A atividade de testes poderia ser vista como destrutiva, ao invés de construtivo durante o processo de engenharia de *software*. Desenvolvedores de *software* são pessoas construtivas e a atividade de teste exige que o desenvolvedor descarte noções preconcebidas da perfeição do *software* que ele acabou de desenvolver e supere um conflito de interesses que ocorre quando erros são descobertos.

Uma série de definições pode servir como objetivos de teste de *software*: (1) a atividade de teste é o processo de executar um programa com a finalidade de encontrar um erro; (2) um bom caso de teste é aquele que tem uma alta probabilidade de descobrir um erro ainda não descoberto; (3) e, um teste bem-sucedido é aquele que revela um erro ainda não descoberto (PRESSMAN, 2011).

A atividade de teste descobrirá erros no *software* caso ela tenha sido conduzida com sucesso. A atividade de teste confirma que as funções de *software* visivelmente estão trabalhando de acordo com as especificações, que os requisitos de desempenho aparentemente foram cumpridos. Os dados compilados quando a atividade de testes é levada a efeito proporcionam uma boa indicação da confiabilidade de *software* e alguma indicação da qualidade do *software* como um todo.

2.2.2 Projeto de Casos de Teste

O projeto de teste de *software* pode vir a ser tão desafiador quanto o projeto inicial do próprio produto. No entanto engenheiros de *software* muitas vezes tratam a

atividade de teste como uma reflexão tardia, desenvolvendo casos de teste que podem "parecer certos", mas que oferecem pouca garantia de estar completos. Os casos de teste devem ser projetados para que se tenha uma maior probabilidade de encontrar a maioria dos erros com uma quantidade mínima de tempo e esforço.

As literaturas (BASTOS et al., 2012) (KOSCIANSKI; SOARES, 2007) (PRESSMAN, 2011) abordam que qualquer que seja o produto trabalhado por engenharia pode vir a ser testado de duas maneiras: *Caixa Preta* (*black-box*) e *Caixa Branca* (*white-box*).

O primeiro teste, Caixa Preta, refere-se aos testes que são realizados nas interfaces do *software*. Esses testes são usados para demonstrar que as funções do *software* são operacionais; que a entrada é adequadamente aceita e a saída é corretamente produzida; que a integridade das informações externas é mantida. Contudo um teste de caixa preta examina alguns aspectos de um *software* sem se preocupar muito com a estrutura lógica interna do *software*.

O segundo, Caixa Branca, trabalha de uma forma minuciosa no exame dos detalhes procedimentais. São testados caminhos lógicos através do *software*, fornecendo-se casos de teste que colocam à prova conjuntos específicos de condições e/ou laços.

2.2.3 Estratégia de Teste de *software*

Os testes de software devem ser empregados de acordo com uma estratégia ou processo. Diversas abordagens podem ser encontradas na literatura (DELAMARO; MALDONADO; JINO, 2007) (BASTOS et al., 2012) (KOSCIANSKI; SOARES, 2007) (SOMMERVILLE, 2011). Apesar das variações existentes, todas concordam que a estratégia de testes deve começar pelo nível de componente, no qual as unidades de código são testadas, e avançar até o nível de sistema, no qual o software e outros elementos de sistemas são testados em conjunto.

Uma estratégia para teste de *software* pode também ser vista no conceito de uma espiral, proposta em (PRESSMAN, 2011). O processo, do ponto de vista procedimental, é composto por quatro etapas que são implementadas sequencialmente, conforme apresentado na Figura 2. Inicialmente, os *testes de unidade* focalizam em cada

componente individualmente, garantindo que ele funcione como um componente. Este teste busca garantir a cobertura completa e a detecção máxima de erro na estrutura de controle de um componente. Em seguida, os componentes devem ser integrados para formar o pacote completo do *software*, sendo assim chamados de *testes de integração*. Estes cuidam de aspectos associados à verificação e construção do programa. Após a integração do *software*, é executada uma série de *testes de validação* sobre os requisitos do produto. Esse teste proporciona a garantia final de que o *software* satisfaz a todos os requisitos informativos, funcionais, comportamentais e de desempenho. Por fim, o *teste de sistema* verifica se todos os elementos se combinam corretamente e se a função/desempenho global do sistema é conseguida.

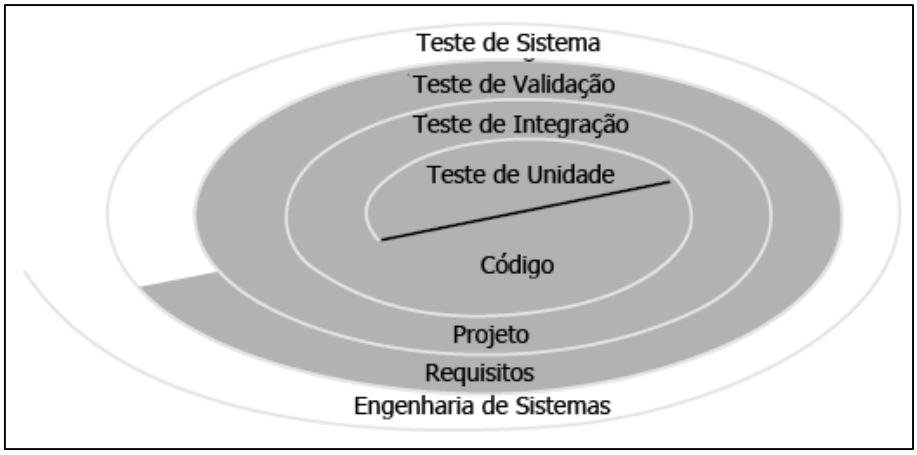

Figura 2: Estrutura de teste. Fonte: adaptado de (PRESSMAN, 2011).

2.3.3.1. Teste de Unidade

O processo de teste de unidades (também chamado de *Teste de Componentes*) é o processo que foca em testar os componentes individuais do sistema, sendo este um processo de teste de defeitos e visa expor defeitos nesses componentes. O mesmo tem como objetivo encontrar defeitos por meio de testes de componentes individuais do *software*, podendo esses componentes ser funções, objetos ou componentes reusáveis (BASTOS et al., 2012) (PEZZÈ; YOUNG, 2008).

Há componentes distintos que podem ser testados nesse estágio como, por exemplo: (1) funções ou métodos individuais de um objeto; (2) classes de objeto com vários atributos e métodos; e (3) componentes compostos que constituem diferentes

objetos ou funções. Tais componentes compostos têm uma interface definida usada para acessar sua funcionalidade.

Os tipos mais simples de componente são funções e métodos individuais e seus testes são um conjunto de chamadas dessas rotinas com diferentes parâmetros de entrada.

Ao iniciar o teste de classes de objeto, deve projetar seus testes de modo que envolvam todas as características do objeto como: (1) testes isolados de todas as operações associadas ao objeto; (2) atribuir e interrogar todos os atributos associados ao objeto; e (3) o exercício do objeto em todos os estados possíveis, quer dizer, que todos os eventos que causam mudanças de estado no objeto devem ser estimulados.

2.2.4 Automação de Testes

Uma vez que até 40% do esforço gasto num projeto de desenvolvimento são dedicados ao teste de *software*, as ferramentas que podem diminuir o tempo de testes são muito valiosas. Sendo o teste de *software* uma fase custosa e trabalhosa do processo de desenvolvimento de *software*, as ferramentas de teste estão entre as primeiras ferramentas de *software* a serem desenvolvidas. Hoje em dia, tais ferramentas oferecem uma variedade de recursos e seu uso pode minimizar significativamente os custos dos testes.

Uma série de categorias de ferramentas de teste são definidas em (MILLER, 1979):

Analisadores estáticos. Esses sistemas de análise de programa suportam a "comprovação" de afirmações estáticas – afirmações fracas sobre a estrutura e o formato de um programa.

Auditores de código. Esses filtros de propósito especiais são usados para verificar a qualidade do *software*, a fim de garantir que ele atenda a padrões mínimos de codificação.

Processadores de asserção. Esses sistemas pré-processadores/pós-processadores são empregados para dizer se as afirmações fornecidas pelo programador, denominadas

asserções, sobre o comportamento de um programa são de fato cumpridas durante as execuções reais do programa.

Geradores de arquivos de teste. Esses processadores geram, e preenchem com valores previamente determinados, arquivos de entrada típicos para programas que estão em teste.

Verificadores de teste. Essas ferramentas medem a cobertura interna dos testes, frequentemente expressa em termos que estão relacionados à estrutura de controle do objeto de teste, e relatam o valor da cobertura ao especialista em garantia da qualidade.

Bancadas de teste (Teste Harnesses). Essa classe de ferramentas apoia o processamento de testes, tornando-o quase indolor, para (1) instalar um programa candidato num ambiente de teste; (2) preencher com dados de entrada; e (3) com o uso de *stubs* o comportamento de módulos subsidiários (subordinados).

Um conjunto integrado de ferramentas que apoiam o processo de teste é chamado de *Workbench* de testes, que além de incluir *framework*s de testes, pode incluir também ferramentas para simular outras partes do sistema a fim de gerar dados de teste.

Algumas dessas ferramentas que podem ser incluídas no *Workbench* são:

Gerenciador de testes. Realiza o gerenciamento da execução de testes de programa. Esse gerenciador de testes mantém o acompanhamento dos dados de teste, resultados esperados e os recursos de programa testados. *framework*s de automação de testes, como JUnit, são exemplos de gerenciadores de testes.

Gerador de dados de teste. Tem o papel de gerar os dados de teste para o programa que será testado. Podendo ser realizado por meio da escolha de dados de um banco de dados ou por meio do uso de padrões para gerar dados aleatórios de maneira correta.

Oráculo. Concentra-se em gerar previsões de resultados de teste esperados. Oráculos podem ser versões anteriores do programa ou protótipos do sistema. O teste *back-to-back* envolve a execução em paralelo do oráculo e do programa que será testado. Saídas distintas são evidenciadas.

Comparador de arquivos. Tem como objetivo comparar os resultados dos testes de programa com os resultados de testes anteriores e relata as diferenças entre eles. São muito utilizados em testes de regressão nos quais são comparados os resultados da execução de versões distintas.

Gerador de relatórios. Fornece recursos de definição e de geração de relatórios para os resultados de teste.

Analisador dinâmico. Conta o número de vezes em que cada declaração foi executada, isso adicionando código ao programa. Após realizar o teste, é gerado um perfil de execução para mostrar a frequência com que cada declaração de programa foi executada.

Simulador. Tipos distintos de simuladores podem ser fornecidos. Simuladores de interface com o usuário são programas dirigidos a scripts que simulam várias interações simultâneas de usuários. Simuladores de alvo simulam a máquina na qual o programa vai ser executado. A utilização de simuladores de E/S significa que o timing de sequencia de transações é repetível.

Muito tempo e esforço são necessários para criar um *Workbench* de teste abrangente. Estes são utilizados apenas quando sistemas de grande porte estão em desenvolvimento. Nesses sistemas, os custos de teste podem chegar a 50% dos custos totais de desenvolvimento, de maneira que é apropriado em termos de custo investir em ferramentas CASE de alta qualidade para apoiar os testes. Porém, ferramentas de teste comerciais podem não estar disponíveis.

A utilização de ferramentas automatizadas durante o processo de teste de *software* está em ascensão. As ferramentas de testes, descendentes da primeira geração, provocarão na maneira como testamos *software* atualmente, mudanças radicais e melhorarão a confiabilidade dos sistemas baseados em computador (PRESSMAN, 2011).

2.3 Qualidade de *Software*

A falta de qualidade nos sistemas de *software* causa grandes prejuízos à economia mundial. Garantir a qualidade de sistemas de *software* é um grande desafio devido à alta complexidade dos produtos e às inúmeras dificuldades relacionadas ao processo de

desenvolvimento, que envolve questões humanas, técnicas, burocráticas, de negócio e políticas (MILLER, 1979).

A não conformidade das necessidades e expectativas significa a falta de qualidade, pois os requisitos são a base através da qual a qualidade é medida. Caso o *software* esteja em conformidade e com os requisitos explícitos, mas deixe de cumprir seus requisitos implícitos, então a qualidade do *software* será comprometida. Para os seguintes autores (CORDEIRO; FREITAS, 2011) (PRESSMAN, 2011) (KOSCIANSKI; SOARES, 2007), esse risco pode afetar a reputação e redução dos lucros da empresa.

De maneira complementar, é imprescindível ressaltar a distinção entre a qualidade voltada para o produto e qualidade voltada para o processo de desenvolvimento. A primeira tradicionalmente se refere à avaliação do software após seu desenvolvimento. Fato esse que levam programadores acreditarem que a qualidade de *software* é algo sobre o qual deve se preocupar depois que o código é gerado, a garantia de qualidade de *software* (SQA, ou *Software Quality Assurance*) é uma atividade universal aplicada em toda a gestão de qualidade. Para o sucesso da SQA é necessário que membros diversos da empresa participem ativamente (CORDEIRO; FREITAS, 2011) (PRESSMAN, 2011).

Uma organização de desenvolvimento de *software* deve adotar procedimentos, métodos e ferramentas de engenharia de *software*, e isso deve ocorrer antes que os procedimentos formais de garantia de qualidade sejam instituídos. Quando combinada com um paradigma eficaz para desenvolvimento de *software*, essa metodologia pode colaborar muito para a melhoria da qualidade de todo o *software* desenvolvido pela organização (KOSCIANSKI; SOARES, 2007).

Na década de 80, foi introduzida uma abordagem da área de qualidade no processo de desenvolvimento de *software*, visando controlar a produção e eliminar a introdução de elementos que possam gerar não conformidade no produto final. Essa abordagem representa os testes de *software* (MILLER, 1979).

Com o passar do tempo, as atividades de teste têm conquistado bastante espaço e autonomia dentro das organizações. Devido à complexidade que os atuais sistemas vêm tendo no seu processo de desenvolvimento, tem-se a necessidade de utilizar

metodologias específicas para que seus processos sejam executados plenamente. É nesse momento em que a qualidade do processo de teste se reflete na qualidade do produto final.

2.4 Qualidade em ambientes de DW

A definição de qualidade de dados trata de dados que estão ausentes, incorretos, inválidos ou até mesmo duplicados, em algum contexto, o que leva à incapacidade destes dados atingirem um valor de qualidade. Uma definição mais ampla é que a qualidade dos dados é alcançada quando a organização utiliza dados que são abrangentes, compreensíveis, consistentes, relevantes e oportunos (IDRIS e AHMAD, 2011). Compreender as principais dimensões de qualidade de dados é o primeiro passo para a melhoria de qualidade dos dados. Para serem processados e interpretados de forma eficaz e eficiente, os dados têm que satisfazer um conjunto de critérios de qualidade. Os dados que satisfazem esses critérios de qualidade são ditos serem de alta qualidade. Dimensões da qualidade dos dados tipicamente incluem precisão, confiabilidade, relevância, consistência, precisão, pontualidade, finura, compreensibilidade, concisão e utilidade (IDRIS e AHMAD, 2011).

Embora o DW seja um ótimo recurso para as empresas, estas encontram obstáculos como os problemas relacionados à qualidade de dados, os quais podem surgir em fases do processo conhecido como ETL (*Extract, Transform and Load*), especialmente no estágio de carga. As principais causas que contribuem para a má qualidade dos dados apontados em [Ranjit and Kawaljeet 2010], são: a) Implementações incorretas ou a má interpretação das estratégias de armazenamento de histórico nas dimensões; b) Falta de tratamento em valores nulos; c) Tratamento incorreto de colunas de auditoria; d) Perda de dados (dados rejeitados) durante o processo de carregamento.

Durante a fase de implementação de um DW, a qualidade de dados é de máxima importância. Assim como no ambiente operacional, os testes podem ser aplicados em um DW para prover a integridade e consistência dos dados. A principal fase onde se deve aplicar processos de testes é na carga dos dados da área auxiliar (*Staging*) para a área de *Data Warehouse* (tabelas de dimensão, fatos, hierarquias e agregados). Podemos

citar como fundamentais pontos a serem contemplados nas fases de testes em DW: testes de unidade e relacionamento em atributos; processamento de dados incorretos, rejeitados e de valores nulos; assim como o tratamento de histórico para os atributos da dimensão. A próxima seção apresenta os principais conceitos de testes de *software*, assim como suas estratégias, projeção de casos de testes e automação.

2.5 Matriz GUT

Segundo (MARSHALL et al., 2011), a matriz GUT é a representação de problemas ou riscos potenciais, através de quantificações que buscam estabelecer prioridades para abordá-los, visando diminuir os impactos.

Esta metodologia propicia a uma empresa ou situação específica a definir suas estratégias e políticas a médio e longo prazo, priorizando as mais importantes levando em consideração alguns parâmetros. Ela tem grande utilidade para a fixação de prioridades na eliminação de problemas, consistindo em separar e priorizar os problemas para fins de análise e posterior solução.

Uma Matriz GUT é feita a partir de parâmetros baseados na gravidade(G), urgência (U) e tendência (T) que são tomados para estabelecer prioridades na eliminação de problemas, orientando assim as decisões mais complexas (GONÇALVES, 2011). Cada um destes aspectos possui um impacto que deve ser analisado e atribuído de acordo com os problemas.

Estes aspectos são definidos da seguinte forma (ABRANTES, 2009):

- Gravidade – G. A atividade é analisada e faz-se a seguinte pergunta: quais as consequências caso a atividade não seja realizada prioritariamente?
- Urgência – U. Faz-se a seguinte pergunta: com que urgência a atividade deve ser realizada?
- Tendência – T. Analisa qual a tendência da não execução imediata da atividade. Pode-se pensar na seguinte pergunta: qual a situação pela não execução imediata da atividade?

Habitualmente atribui-se valores entre 1 e 5, a cada uma das dimensões (G.U.T), correspondendo o 5 à maior intensidade e o 1 à menor. Os valores obtidos para o G, U e T, serão multiplicados a fim de se obter um valor para cada problema ou fator de risco estudado (TRISTÃO, 2011). Os problemas ou fatores de risco que obtiverem maior pontuação serão tratados prioritariamente (MARSHALL et al., 2011).

O Quadro 1 mostra um exemplo básico da estrutura de uma matriz GUT.

Quadro 1: EXEMPLO BÁSICO DE MATRIZ GUT (ABRANTES, 2009).

Valor	**Gravidade** (Consequências se nada for feito)	**Urgência** (Prazo para tomada de decisão)	**Tendência** (Proporção de problema no futuro)	**Valor Total G x U x T**
5	Os prejuízos ou dificuldades são extremamente graves	É necessária uma ação imediata	Se nada for feito, o agravamento da situação é imediata	5x5x5 = 125
4	Muito grave	Com alguma urgência	Vai piorar no curto prazo	4x4x4 = 64
3	Graves	O mais cedo possível	Vai piorar no médio prazo	3x3x3 = 27
2	Pouco grave	Pode esperar um pouco	Vai piorar no longo prazo	2x2x2 = 8
1	Sem gravidade	Não tem pressa	Não vai piorar (e pode até melhorar)	1x1x1 = 1

A matriz GUT foi utilizada, neste trabalho, como forma de medir o grau de criticidade dos erros encontrados nas cargas de dados geradas pelos procedimentos criados, de forma manual e automática.

CAPÍTULO 3

3 *FRAMEWORK* DE TESTES

Os problemas com a qualidade de dados apontados em (RANJIT; KAWALJEET SINGH, 2010), principalmente na fase de carga, assim como a identificação da falta de facilidades para testes de unidade em ferramentas ETL foi a principal motivação para propor uma abordagem baseada em um *framework* de testes de unidade para rotinas de carga em ambientes de DW. Outra motivação foi a comprovação na literatura da necessidade de diferentes abordagens de testes para as diferentes fases do processo de *Data Warehousing*, diante da impossibilidade ou dificuldade no emprego das técnicas tradicionais (GOLFARELLI; RIZZI, 2009) (ELGAMAL, 2013) (DESHPANDE, 2013).

Ao contrário das unidades de programa tradicionais, as rotinas de carga são alvos de constantes mudanças mesmo após a implantação em produção. Novas formas de análise dos dados solicitadas pelos gerentes e analistas de negócio, cruzamento de informações com novas fontes externas, e inclusões ou alterações nos esquemas de dados operacionais são alguns dos cenários que exigem a existência de testes de regressão. Nesse capítulo, são descritos os Casos de Testes definidos para elaboração do *framework* e utilização dos experimentos. Também é apresentado o projeto do *framework* de testes de unidade.

3.1 Projeto de Casos de Testes

As rotinas de carga para o ambiente de DW são bastante discutidas em (COLAÇO, 2004) (KIMBALL, 2002) (KIMBALL, 2004) (KIMBALL, 2008). Abordagens alternativas para a carga de dimensões podem ser encontradas em (SANTOS; BELO, 2011). Algoritmos para as rotinas de carga para os vários tipos de comportamento de dimensões podem ser encontrados em (SANTOS; COSTA; NASCIMENTO, COLAÇO, 2012a) (SANTOS; NASCIMENTO, 2012b). Categorias de

casos de testes para as rotinas ETL são apontadas em (ELGAMAL, 2013) (COOPER; ARBUCKLE, 2002). Esse material serviu de base para a elaboração das categorias de casos de testes a serem considerados pelo *framework*.

As seguintes categorias são contempladas pelo *framework*:

- Testes de unicidade e relacionamento;
- Número de registros entre fonte e destino;
- Transformações entre fonte e destino;
- Processamento de dados incorretos ou rejeitados;
- Processamento de valores nulos;
- Comportamentos tipo 1[2] e tipo 2[3] para atributos de dimensões;
- Abordagens híbridas para o tratamento de histórico em dimensões (tipo 4 e tipo 6).

Uma vez que a literatura não apresenta a definição de casos de testes para rotinas de carga, foi necessário definir casos de testes a serem tratados pelo *framework*.

3.1.1 Caso de Teste 01 (TC_001)

O Quadro 1 apresenta a descrição para a primeira carga na dimensão com atributos tipo 1.

Quadro 2: Descrição detalhada do Caso de Teste 01.

Identificador	**TC_001**
Descrição	Teste de primeira carga para dimensão com atributos do tipo 1.
Objetivo	Verificar se o número de registros na dimensão corresponde ao número de registros da tabela auxiliar.
Pré-condições	a) Dimensão vazia; e, b) Registros incluídos na tabela auxiliar (todos os registros devem apresentar valores para todos os atributos).
Procedimentos	a) Eliminar linhas da tabela auxiliar; b) Eliminar linhas da tabela de dimensão; c) Incluir registros na tabela auxiliar; e,

[2] O tipo 1 é a abordagem de reescrita, sem salvaguardar os dados passados (SANTOS e BELO, 2011).

[3] O tipo 2, cria-se um novo registro para dado alterado e o registro antigo é mantido como histórico (SANTOS e BELO, 2011).

	d) Executar o procedimento de carga.
Resultado Esperado	O número de registros na dimensão corresponde ao número de registros incluídos na tabela auxiliar.

- **Exemplo:**

Para exemplificar a descrição do Caso de Teste 01 (Quadro 2), os quadros a seguir demonstram como seriam o carregamento inicialmente para a tabela auxiliar (Quadro 3) e os dados carregados na dimensão (Quadro 4) após a carga dos dados.

Quadro 3: Registros na tabela auxiliar de funcionário.

COD_FUNCIONARIO	NOME	ENDERECO	CARGO	DATA_CARGA
1	JOÃO	RUA A	ENGENHEIRO	02/02/2016
2	JOSÉ	RUA B	ARQUITETO	02/02/2016
3	MARIA	RUA C	ENFERMEIRA	02/02/2016

Quadro 4: Registros na dimensão de funcionários após a execução do procedimento de carga.

ID_FUNC	COD_FUNCIONARIO	NOME	ENDERECO	CARGO
1	1	JOÃO	RUA A	ENGENHEIRO
2	2	JOSÉ	RUA B	ARQUITETO
3	3	MARIA	RUA C	ENFERMEIRA

3.1.2 Caso de Teste 02 (TC_002)

O Quadro 5 descreve o teste de segunda carga para uma dimensão com atributos do tipo 1.

Quadro 5: Descrição detalhada do Caso de Teste 02.

Identificador	**TC_002**
Descrição	Teste de segunda carga para dimensão com atributos do tipo 1.
Objetivo	Verificar se cargas consecutivas sem alterações no ambiente operacional ocasionam a criação de registros incorretos na dimensão.
Pré-condições	a) O resultado do teste TC_001 está ok; b) Dimensão vazia; e, c) Registros incluídos na tabela auxiliar (todos os registros devem apresentar valores para todos os atributos).

Procedimentos	a) Eliminar linhas da tabela auxiliar; b) Eliminar linhas da tabela de dimensão; c) Incluir registros na tabela auxiliar; d) Executar o procedimento de carga; e, e) Executar o procedimento de carga novamente.
Resultado Esperado	O número de registros na dimensão corresponde ao número de registros incluídos na tabela auxiliar.

O exemplo deste caso de teste segue o mesmo que o anterior (TC_001). O objetivo deste é mostrar que os dados não sofrerão alterações ou criação de novos registros.

3.1.3 Caso de Teste 03 (TC_003)

O Quadro 6 apresenta o caso de teste de unicidade para dimensão com atributos do tipo 1, que tem como objetivo verificar se os dados contidos na área de *staging* (auxiliar) são os mesmos contidos na dimensão.

Quadro 6: Descrição detalhada do Caso de Teste 03.

Identificador	**TC_003**
Descrição	Teste de unicidade para dimensão com atributos do tipo 1.
Objetivo	Verificar se todo registro na área auxiliar (uma determinada chave natural) possui uma e somente uma correspondência na dimensão.
Pré-condições	a) O resultado do teste TC_001 e TC_002 estão ok; b) Dimensão vazia; e, c) Registros incluídos na tabela auxiliar (todos os registros devem apresentar valores para todos os atributos).
Procedimentos	a) Eliminar linhas da tabela auxiliar; b) Eliminar linhas da tabela de dimensão; c) Incluir registros na tabela auxiliar; e, d) Executar o procedimento de carga.
Resultado Esperado	Cada chave natural de registro na área auxiliar possui um e somente um registro na dimensão.

- **Exemplo:**

O resultado final do caso de teste deve seguir como nos quadros a seguir (Quadro 7 e Quadro 8). Não diferente do que foi apresentado em TC_001, mas importante destacar que no segundo quadro (Quadro 8) demonstra que para cada registro adicionado na tabela auxiliar há um e somente um registro na dimensão.

Quadro 7: Registros na tabela auxiliar de funcionário.

COD_FUNCIONARIO	NOME	ENDERECO	CARGO	DATA_CARGA
1	JOÃO	RUA A	ENGENHEIRO	02/02/2016
2	JOSÉ	RUA B	ARQUITETO	02/02/2016
3	MARIA	RUA C	ENFERMEIRA	02/02/2016

Quadro 8: Registros na dimensão de funcionário após a execução do procedimento de carga.

ID_FUNC	COD_FUNCIONARIO	NOME	ENDERECO	CARGO
1	1	JOÃO	RUA A	ENGENHEIRO
2	2	JOSÉ	RUA B	ARQUITETO
3	3	MARIA	RUA C	ENFERMEIRA

3.1.4 Caso de Teste 04 (TC_004)

O Quadro 9 descreve como deve ocorrer os testes de integridade para dimensões com atributos do tipo 1. Além de ter como objetivo verificar se os registros sofrem ou não alterações após o procedimento de carga de dados.

Quadro 9: Descrição detalhada do Caso de Teste 04.

Identificador	**TC_004**
Descrição	Teste de integridade para dimensão com atributos do tipo 1.
Objetivo	Verificar se os valores dos atributos de cada registro da área auxiliar, caso não sofram transformações, correspondem aos valores dos atributos na dimensão. O objetivo é verificar se não houve transformações inesperadas, truncamento ou perda de precisão.
Pré-condições	a) O resultado dos teste TC_001, TC_002 e TC_003 estão OK; b) Dimensão vazia; e, c) Registros incluídos na tabela auxiliar (todos os registros devem apresentar valores para todos os atributos).
Procedimentos	a) Eliminar linhas da tabela auxiliar; b) Eliminar linhas da tabela de dimensão; c) Incluir registros na tabela auxiliar; e, d) Executar o procedimento de carga.
Resultado Esperado	Todo valor de atributo da área auxiliar, caso não sofra transformações, permanece inalterado no atributo correspondente da dimensão.

- **Exemplo:**

Como pode ser visto nos quadros a seguir (Quadro 10 e Quadro 11) houve um problema de integridade na carga dos dados para a dimensão, uma vez que o dado do funcionário, cujo código é 2, está com o endereço errado, quando deveria ser 'RUA B'. Caso semelhante ocorre com o funcionário de código 3. Neste o atributo nome que deveria vir como 'JOSÉ e está 'JOS3'.

Quadro 10: Registros na tabela auxiliar de funcionário para uma segunda carga.

COD_FUNCIONARIO	NOME	ENDERECO	CARGO	DATA_CARGA
1	IGOR	RUA A	ENGENHEIRO	02/02/2016
2	LUCAS	RUA B	ARQUITETO	02/02/2016
3	JOSEFA	RUA C	ENFERMEIRA	02/02/2016

Quadro 11: Registros na dimensão de funcionário após a execução do procedimento de carga com problema de integridade dos dados.

ID_FUNC	COD_FUNCIONARIO	NOME	ENDERECO	CARGO
1	1	IGOR	RUA A	ENGENHEIRO
2	2	LUCAS	RUA	ARQUITETO
3	3	JOS3	RUA C	ENFERMEIRA

3.1.5 Caso de Teste 05 (TC_005)

O Caso de Teste 05 é descrito no Quadro 12. Neste há o primeiro caso de teste para carga de atributos tipo 2 de uma dimensão com o objetivo de averiguar se os registros não sofreram alterações e se os atributos de gerenciamento de históricos foram carregados corretamente.

Quadro 12: Descrição detalhada do Caso de Teste 05.

Identificador	**TC_005**
Descrição	Teste de primeira carga para dimensão com atributos do tipo 2.
Objetivo	Verificar se o número de registros na dimensão corresponde ao número de registros da tabela auxiliar. Além disso, conferir se os registros foram incluídos com informações corretas para os

	atributos de gerenciamento de histórico: data de início, data final e *flag* corrente.
Pré-condições	a) Dimensão vazia; e, b) Registros incluídos na tabela auxiliar (todos os registros devem apresentar valores para todos os atributos).
Procedimentos	a) Eliminar linhas da tabela auxiliar; b) Eliminar linhas da tabela de dimensão; c) Incluir registros na tabela auxiliar; e, d) Executar o procedimento de carga.
Resultado Esperado	O número de registros na dimensão corresponde ao número de registros incluídos na tabela auxiliar. Os registros foram incluídos com informações corretas para os atributos de gerenciamento de histórico. Data início igual à data da carga, data fim igual a *null* e fl_corrente igual a 's' (sim).

- **Exemplo:**

Os quadros abaixo (Quadro 13 e Quadro 14) definem o exemplo que deve ocorrer para o TC_005. Os atributos, neste exemplo, que correspondem ao tipo 2 da dimensão (Quadro 14) são ENDERECO e CARGO. Logo, nota-se que os mesmos foram carregados corretamente provenientes da tabela auxiliar (Quadro 13). Além disso, os atributos (data_inicio, data_fim e fl_corrente.) que gerenciam o histórico tipo 2 numa dimensão foram corretamente preenchidos. A data de início recebeu o valor '02/02/2016', a data de fim recebe o valor nulo na dimensão, e por fim, o *flag* corrente o valor 'S'.

Quadro 13: Registros na tabela auxiliar com os dados de funcionários.

COD_FUNC	NOME	ENDERECO	CARGO	DATA_CARGA
1	JOÃO	RUA A	ENGENHEIRO	02/02/2016
2	JOSÉ	RUA B	ARQUITETO	02/02/2016
3	MARIA	RUA C	ENFERMEIRA	02/02/2016

Quadro 14: Registros na dimensão contendo atributos do tipo 2

ID_FUNC	COD_FUNC	NOME	ENDERECO	CARGO	DT_INICIO	DT_FIM	FL_CORRENTE
1	1	JOÃO	RUA A	ENGENHEIRO	02/02/2016	*NULL*	S
2	2	JOSÉ	RUA B	ARQUITETO	02/02/2016	*NULL*	S
3	3	MARIA	RUA C	ENFERMEIRA	02/02/2016	*NULL*	S

3.1.6 Caso de Teste 06 (TC_006)

A descrição detalhada do Caso de Teste 06 encontra-se no quadro abaixo (Quadro 15). O TC_006 verifica se o(s) valor(es) de atributos do tipo 1 sofrem suas devidas alterações corretamente sem influenciar o histórico dos atributos tipo 2 de uma dimensão.

Quadro 15: Descrição detalhada do Caso de Teste 06.

Identificador	**TC_006**
Descrição	Teste de modificação de dados em atributos tipo 1 para dimensão que também contenha atributos do tipo 2.
Objetivo	Verificar se o valor de atributo de tipo 1 na dimensão foi devidamente alterado.
Pré-condições	a) Os resultados dos teste TC _001 e TC_002 estão ok; b) Dimensão vazia; e, c) Registros incluídos na tabela auxiliar (todos os registros devem apresentar valores para todos os atributos).
Procedimentos	a) Eliminar linhas da tabela auxiliar; b) Eliminar linhas da tabela de dimensão; c) Incluir registros na tabela auxiliar; d) Executar o procedimento de carga; e) Alterar, para um registro na área auxiliar, os valores de todos os atributos com comportamento tipo 1; e, f) executar o procedimento de carga.
Resultado Esperado	Os valores dos atributos do tipo 1 no registro que corresponde à alteração efetuada na área auxiliar foram devidamente atualizados.

- **Exemplo:**

Com base no exemplo do Caso de Teste anterior (CT_005), os quadros Quadro **16** e Quadro **17** apresentam um segundo momento para o procedimento de carga. No Quadro 16 há novos registros para a data de carga ‘03/03/2016’. Nota-se que o registro, de COD_FUNC 2, no atributo de tipo 1 (NOME) sofreu alteração em relação à carga anterior (‘02/02/2016’), que era ‘JOSÉ’ e agora é ‘CARLOS’. Após o procedimento de carga para a dimensão, o valor desse atributo é, e somente ele, alterado corretamente.

Quadro 16: Alteração de atributo tipo 1 (nome) na tabela auxiliar numa segunda data de carga.

COD_FUNC	NOME	ENDERECO	CARGO	DATA_CARGA
1	JOÃO	RUA A	ENGENHEIRO	02/02/2016

2	JOSÉ	RUA B	ARQUITETO	02/02/2016
3	MARIA	RUA C	ENFERMEIRA	02/02/2016
1	JOÃO	RUA A	ENGENHEIRO	03/03/2016
2	CARLOS	RUA B	ARQUITETO	03/03/2016
3	MARIA	RUA C	ENFERMEIRA	03/03/2016

Quadro 17: Atributo tipo 1 alterado na dimensão após a execução do procedimento de carga.

ID_FUNC	COD_FUNC	NOME	ENDERECO	CARGO	DT_INICIO	DT_FIM	FL_CORRENTE
1	1	JOÃO	RUA A	ENGENHEIRO	02/02/2016	*NULL*	S
2	2	CARLOS	RUA B	ARQUITETO	02/02/2016	*NULL*	S
3	3	MARIA	RUA C	ENFERMEIRA	02/02/2016	*NULL*	S

3.1.7 Caso de Teste 07 (TC_007)

O Quadro 18, que descreve o Caso de Teste 07, apresenta os testes de modificação para os atributos do tipo 2 e como seus atributos gerenciadores de históricos devem se comportar.

Quadro 18: Descrição detalhada do Caso de Teste 07.

Identificador	**TC_007**
Descrição	Teste de modificação de dados para dimensão com atributos do tipo 2.
Objetivo	Verificar se a alteração de valor foi realizada da forma correta: adição de novo registro na dimensão e ajuste dos atributos para gerenciamento de histórico (data de início, data final e *flag* corrente). Esse teste deve ser repetido, de forma individual, para cada atributo com comportamento tipo 2.
Pré-condições	a) O resultado dos testes TC_005 está ok; b) Dimensão vazia; e, c) Registros incluídos na tabela auxiliar (todos os registros devem apresentar valores para todos os atributos).
Procedimentos	a) Eliminar linhas da tabela auxiliar; b) Eliminar linhas da tabela de dimensão; c) Incluir registros na tabela auxiliar; d) Executar o procedimento de carga; e) Alterar, para um registro na área auxiliar, o valor dos atributos

	com comportamento tipo 2; e, f) Executar o procedimento de carga.
Resultado Esperado	O número de registros na dimensão é igual ao número de registros anterior à modificação + 1. O registro na dimensão, correspondente à alteração, manteve os valores originais e apenas o atributo data fim foi atualizado para a data de carga. Há uma adição de um novo registro na dimensão correspondente à alteração realizada na área auxiliar. O novo registro possui os seguintes valores para os atributos de gerenciamento de histórico: data de início igual à data da carga, data final sem valor (*null*) e *flag* corrente igual a 'S' (sim).

- **Exemplo:**

Os quadros Quadro **19** e Quadro **20** exemplificam o Caso de Teste 07. No primeiro há duas cargas, em datas diferentes, para uma tabela auxiliar de funcionários. Para a segunda carga ('03/03/2016') foi alterado o valor do atributo CARGO de 'ENFERMEIRA' para 'ANALISTA'. Isso será suficiente para gerar um novo registro na tabela de dimensão após a segunda carga (Quadro 20).

Quando ocorre a execução do procedimento para uma segunda data na dimensão, cujo atributo tipo 2 houve uma modificação, há a modificação dos atributos de gerenciamento de histórico e um novo registro é adicionado com os valores mais atuais (Quadro 21).

Quadro 19: Registros na tabela auxiliar com duas datas de carga e alteração no atributo CARGO do tipo 2.

COD_FUNC	NOME	ENDERECO	CARGO	DATA_CARGA
1	JOÃO	RUA A	ENGENHEIRO	02/02/2016
2	JOSÉ	RUA B	ARQUITETO	02/02/2016
3	MARIA	RUA C	ENFERMEIRA	02/02/2016
1	JOÃO	RUA A	ENGENHEIRO	03/03/2016
2	CARLOS	RUA B	ARQUITETO	03/03/2016
3	MARIA	RUA C	ANALISTA	03/03/2016

Quadro 20: Registros na dimensão após a execução do procedimento de duas cargas.

ID_FUNC	COD_FUNC	NOME	ENDERECO	CARGO	DT_INICIO	DT_FIM	FL_CORRENTE
1	1	JOÃO	RUA A	ENGENHEIRO	02/02/2016	*NULL*	S
2	2	CARLOS	RUA B	ARQUITETO	02/02/2016	*NULL*	S
3	3	MARIA	RUA C	ENFERMEIRA	02/02/2016	03/03/2016	N
4	3	MARIA	RUA C	ANALISTA	03/03/2016	*NULL*	S

3.1.8 Caso de Teste 08 (TC_008)

Semelhante ao TC_007, o Caso de Teste 08 verifica também as modificações para os atributos tipo 2 de uma dimensão. Porém, neste caso de teste a validação se dará por meio da verificação da alteração em todos os atributos tipo 2 de um registro, por vez.

Quadro 21: Descrição detalhada do Caso de Teste 08.

Identificador	**TC_008**
Descrição	Teste de modificação de dados para dimensão com atributos do tipo 2. As modificações devem ser em um registro para todos os atributos do tipo 2.
Objetivo	Verificar se a alteração de valor foi realizada da forma correta: adição de novo registro na dimensão e ajuste dos atributos para gerenciamento de histórico.
Pré-condições	a) O resultado do teste TC_007 está ok; b) Dimensão vazia; e, c) Registros incluídos na tabela auxiliar (todos os registros devem apresentar valores para todos os atributos).
Procedimentos	a) Eliminar linhas da tabela auxiliar; b) Eliminar linhas da tabela de dimensão; c) Incluir registros na tabela auxiliar; d) Executar o procedimento de carga; e) Alterar, para um registro na área auxiliar, os valores de todos os atributos com comportamento tipo 2; e, f) Executar o procedimento de carga.
Resultado Esperado	O número de registros na dimensão é igual ao número de registros anterior à modificação + 1. O registro na dimensão, correspondente à alteração, manteve os valores originais e apenas o atributo data fim foi atualizado para a data de carga. Há uma adição de um novo registro na dimensão correspondente à alteração realizada na área auxiliar. O novo registro possui os seguintes valores para os atributos de gerenciamento de histórico: data de início igual à data da carga, data final sem valor (*null*) e *flag* corrente igual a 'S' (sim).

- **Exemplo:**

Este exemplo é bem semelhante ao CT_007, a diferença como pode ser vista nos quadros a seguir é que o registro, cujo COD_FUNC é igual a 3, houve alteração em todos os seus atributos com comportamento tipo 2. Essa mudança na tabela auxiliar (Quadro 22) reflete no Quadro 23 onde pode ser visto a adição de um novo registro (ID_FUNC igual a 4) com todas as modificações dos atributos.

Quadro 22: Registros na tabela auxiliar com duas datas de carga e alteração nos atributos do tipo 2 em um registro.

COD_FUNC	NOME	ENDERECO	CARGO	DATA_CARGA
1	JOÃO	RUA A	ENGENHEIRO	02/02/2016
2	JOSÉ	RUA B	ARQUITETO	02/02/2016
3	MARIA	RUA C	ENFERMEIRA	02/02/2016
1	JOÃO	RUA A	ENGENHEIRO	03/03/2016
2	CARLOS	RUA B	ARQUITETO	03/03/2016
3	MARIA	RUA ABC	ANALISTA	03/03/2016

Quadro 23: Registros na dimensão após a execução do procedimento de duas cargas e alteração em um todos os atributos do tipo 2 em um registro.

ID_FUNC	COD_FUNC	NOME	ENDERECO	CARGO	DT_INICIO	DT_FIM	FL_CORRENTE
1	1	JOÃO	RUA A	ENGENHEIRO	02/02/2016	*NULL*	S
2	2	CARLOS	RUA B	ARQUITETO	02/02/2016	*NULL*	S
3	3	MARIA	RUA C	ENFERMEIRA	02/02/2016	03/03/2016	N
4	3	MARIA	RUA ABC	ANALISTA	03/03/2016	*NULL*	S

3.1.9 Caso de Teste 09 (TC_009)

O Caso de Teste 09 descreve como os atributos do tipo 2, que podem apresentar valores nulos, sejam modificados corretamente.

Quadro 24: Descrição detalhada do Caso de Teste 09.

Identificador	**TC_009**
Descrição	Teste de modificação de dados para dimensão com atributos do tipo 2. As modificações nos atributos podem apresentar valores nulos iniciais.
Objetivo	Realizar teste de forma repetida e individual para cada atributo com comportamento tipo 2 que possa ser nulo na dimensão.
Pré-condições	a) O resultado do teste TC_008 está ok; b) Dimensão vazia; e, c) Registros incluídos na tabela auxiliar (um registro com valor nulo para um atributo com comportamento tipo 2 e que aceita a restrição de nulidade *null* na dimensão).
Procedimentos	a) Eliminar linhas da tabela auxiliar; b) Eliminar linhas da tabela de dimensão; c) Incluir registros na tabela auxiliar; e, d) Executar o procedimento de carga.
Resultado Esperado	O número de registros na dimensão é igual ao número de registros da tabela auxiliar. Os registros da dimensão não são modificados. Novos registros não são adicionados.

- **Exemplo:**

Para exemplificar o Caso de Teste 09 foram definidas três datas de carga para a tabela auxiliar (Quadro 25). Na última carga o registro de código 3 recebe a ausência de valor para o atributo CARGO.

Após executar, nas três datas, o procedimento de carga para a dimensão, nota-se que o registro de ID_FUNC igual a 4 recebe *null* para o atributo cargo, porém isso não gera um novo registro na dimensão, como mostra o Quadro 26.

Quadro 25: Registos na tabela auxiliar com carga para três datas.

COD_FUNC	NOME	ENDERECO	CARGO	DATA_CARGA
1	JOÃO	RUA A	ENGENHEIRO	02/02/2016
2	JOSÉ	RUA B	ARQUITETO	02/02/2016
3	MARIA	RUA C	ENFERMEIRA	02/02/2016
1	JOÃO	RUA A	ENGENHEIRO	03/03/2016
2	CARLOS	RUA B	ARQUITETO	03/03/2016
3	MARIA	RUA ABC	ANALISTA	03/03/2016
1	JOÃO	RUA A	ENGENHEIRO	04/04/2016

2	CARLOS	RUA B	ARQUITETO	04/04/2016
3	MARIA	RUA ABC	*NULL*	04/04/2016

Quadro 26: Registros na dimensão após a carga das três datas e atributo tipo 2 com valor nulo.

ID_FUNC	COD_FUNC	NOME	ENDERECO	CARGO	DT_INICIO	DT_FIM	FL_CORRENTE
1	1	JOÃO	RUA A	ENGENHEIRO	02/02/2016	*NULL*	S
2	2	CARLOS	RUA B	ARQUITETO	02/02/2016	*NULL*	S
3	3	MARIA	RUA C	ENFERMEIRA	02/02/2016	03/03/2016	N
4	3	MARIA	RUA ABC	*NULL*	03/03/2016	*NULL*	S

3.1.10 Caso de Teste 10 (TC_010)

O último Caso de Teste, TC_010, apresenta modificações num atributo do tipo 1 quando há histórico no armazenamento de atributos do tipo 2. Logo, deseja verificar se a modificação irá se propagar em todos os registros na dimensão.

Quadro 27: Descrição detalhada do Caso de Teste 10.

Identificador	**TC_010**
Descrição	Teste de modificação de dados para dimensão com atributos do tipo 1 em dimensões com atributos do tipo 2 com histórico armazenado.
Objetivo	Verificar se uma alteração em um atributo do tipo 1 é propagada para todas as versões de registro.
Pré-condições	a) O resultado do teste TC_007 e TC_009 estão ok; b) Dimensão vazia; e, c) Registros incluídos na tabela auxiliar.
Procedimentos	a) Eliminar linhas da tabela auxiliar; b) Eliminar linhas da tabela de dimensão; c) Incluir registros na tabela auxiliar; d) Executar o procedimento de carga; e) Alterar, para um registro na área auxiliar, o valor de um atributo com comportamento tipo 2; f) Executar o procedimento de carga; g) Alterar, para o mesmo registro anteriormente modificado (chave natural) na área auxiliar, os valores dos atributos com comportamento tipo 1; e, h) Executar o procedimento de carga.
Resultado	Os valores dos atributos do tipo 1 em todos os registros que

Esperado	correspondem à alteração efetuada na área auxiliar foram devidamente atualizados. (todas as versões).

- **Exemplo:**

O Quadro 28 apresenta a tabela auxiliar com a carga para três datas diferentes. A última, além das modificações explicadas nos exemplos anteriores, contém uma modificação no atributo nome do tipo 1. Após a execução do procedimento de carga das três datas para dimensão, nota-se, no Quadro 29, que o valor 'JOSEFA' do atributo nome é refletido para todos os registros cujo COD_FUNC é igual a 3.

Quadro 28: Registos na tabela auxiliar com valor alterado para atributo tipo 1 numa terceira data de carga.

COD_FUNC	NOME	ENDERECO	CARGO	DATA_CARGA
1	JOÃO	RUA A	ENGENHEIRO	02/02/2016
2	JOSÉ	RUA B	ARQUITETO	02/02/2016
3	MARIA	RUA C	ENFERMEIRA	02/02/2016
1	JOÃO	RUA A	ENGENHEIRO	03/03/2016
2	CARLOS	RUA B	ARQUITETO	03/03/2016
3	MARIA	RUA ABC	ANALISTA	03/03/2016
1	JOÃO	RUA A	ENGENHEIRO	04/04/2016
2	CARLOS	RUA B	ARQUITETO	04/04/2016
3	JOSEFA	RUA ABC	*NULL*	04/04/2016

Quadro 29: Registros na dimensão após a carga das três datas e modificação nos registros com atributo tipo 1 modificado.

ID_FUNC	COD_FUNC	NOME	ENDERECO	CARGO	DT_INICIO	DT_FIM	FL_CORRENTE
1	1	JOÃO	RUA A	ENGENHEIRO	02/02/2016	*NULL*	S
2	2	CARLOS	RUA B	ARQUITETO	02/02/2016	*NULL*	S
3	3	JOSEFA	RUA C	ENFERMEIRA	02/02/2016	03/03/2016	N

4	3	JOSEFA	RUA ABC	*NULL*	03/03/2016	*NULL*	S

3.2 Projeto do *framework*

O *framework* proposto neste trabalho, denominado de FTUnit, tem como objetivo realizar testes de unidade nos procedimentos de cargas de dados num ambiente de *Data Warehouse*. Utilizando o padrão de projeto arquitetural MCV(*Model View Controll)* (PRESSMAN, 2011), o *framework* foi desenvolvido, de forma iterativa e incremental, na linguagem C# para ser executado em ambientes *Desktops.* O *framework* está disponível para *download* em <http://ftunit.wordpress.com/>.

Esta seção irá apresentar os artefatos da UML (*Unified Modeling Language*) que foram projetados para a concepção do *framework*, são eles: casos de uso, visão lógica, diagrama de classe, diagrama de sequência e o modelo orientado a objetos.

3.2.1 Visão de Casos de Uso

Nas primeiras etapas de análise e *design* da ferramenta, foram definidos os seguintes casos de uso para desenvolvimento: CSU01- Manter dimensão; CSU02 – Manter Tabela Auxiliar; CSU03 – Manter Procedimento; CSU04 – Manter Realizar Testes; e, CSU05 – Relatórios. Os mesmos são descritos logo após o diagrama de caso de uso apresentado na Figura 3.

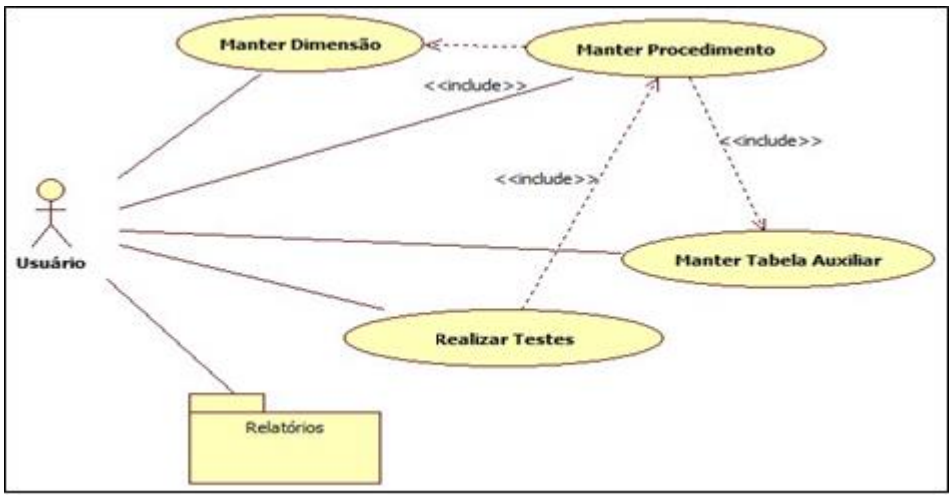

Figura 3: Diagrama de Casos de Uso do *framework*.

- **CSU01 – Manter Dimensão:** o usuário precisa inserir o nome do banco, o esquema e o nome da dimensão. Em seguida, deve detalhar as informações de cada atributo dessa dimensão;

- **CSU02 – Manter Tabela Auxiliar:** o usuário precisa inserir o nome do banco, o esquema e o nome da tabela auxiliar. Logo após, deve detalhar as informações de cada atributo dessa tabela auxiliar;
- **CSU03 – Manter Procedimento:** o usuário precisa inserir o nome do banco, o esquema, o nome do procedimento e, também, inserir uma dimensão e uma tabela auxiliar. Posteriormente, deve informar os atributos de ligação entre a dimensão e a tabela auxiliar;
- **CSU04 –Realizar Testes:** o usuário informa qual procedimento deseja testar e seleciona os casos de testes que deseja executar; e,
- **CSU05 – Relatórios:** contém os relatórios que são gerados pela ferramenta para visualização de resultados e logs.

3.2.2 Visão Lógica

A visão lógica do *framework* é composta principalmente por três pacotes:

- *Model*: Esse pacote representa a implementação da parte lógica do domínio da aplicação. Aqui serão armazenadas classes que representarão entidades da aplicação e classes responsáveis pela persistência dos dados da aplicação;
- *Views*: Nesse pacote são armazenados os componentes que fazem parte da interface gráfica da aplicação;
- *Controller*: Nesse pacote são armazenados os componentes de controle. Esses componentes são responsáveis por integrar os *Models* com as *Views*, de acordo com a interação com usuário.

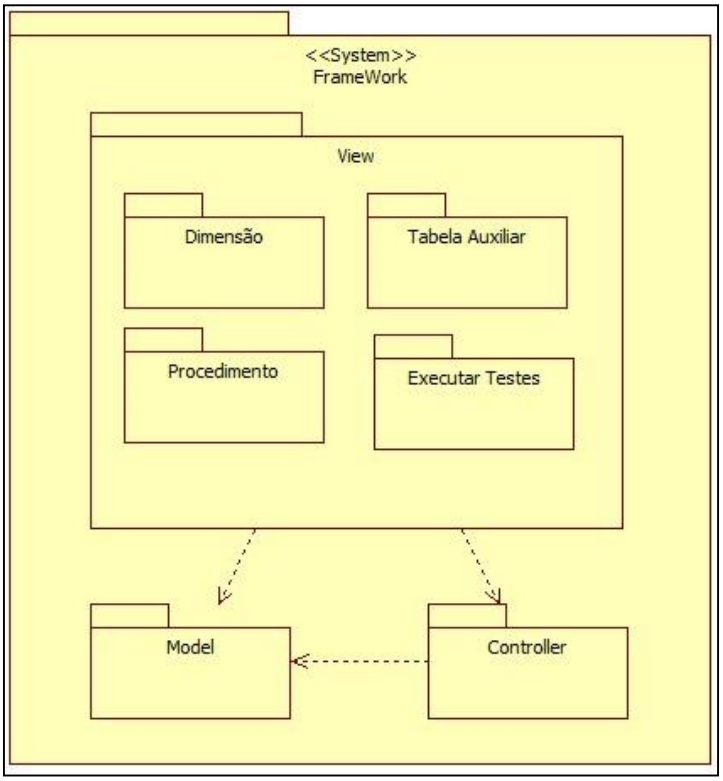

Figura 4: Diagrama de Pacotes do *framework*.

3.2.3 Diagrama de Classes

Dentre os diagramas de classes criados para o desenvolvimento do *framework*, aqui será apresentado apenas o UC04- Realizar Testes. Este é fundamental para entendimento de como o *framework* gerencia o procedimento (que contém a tabela auxiliar e dimensão), assim como os casos de testes a serem executadas pela mesma. Sendo assim, o diagrama de classe da Figura 5 modela as classes, com métodos e relacionamentos a serem utilizados.

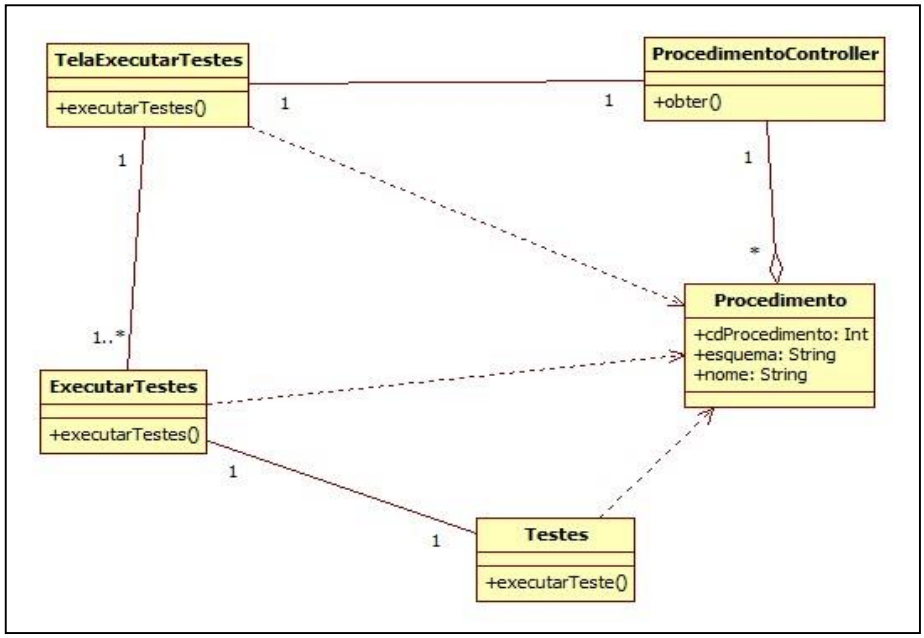

Figura 5: Diagrama de Classe do caso de uso UC04-Realizar Testes.

3.2.4 Diagrama de Sequência

Dando continuidade à modelagem, a seguir (Figura 6) é apresentado o diagrama de sequência do caso de uso Manter Dimensão. Nele há a sequência de passos que são necessários para que os Casos de Testes possam ser executados no *framework*.

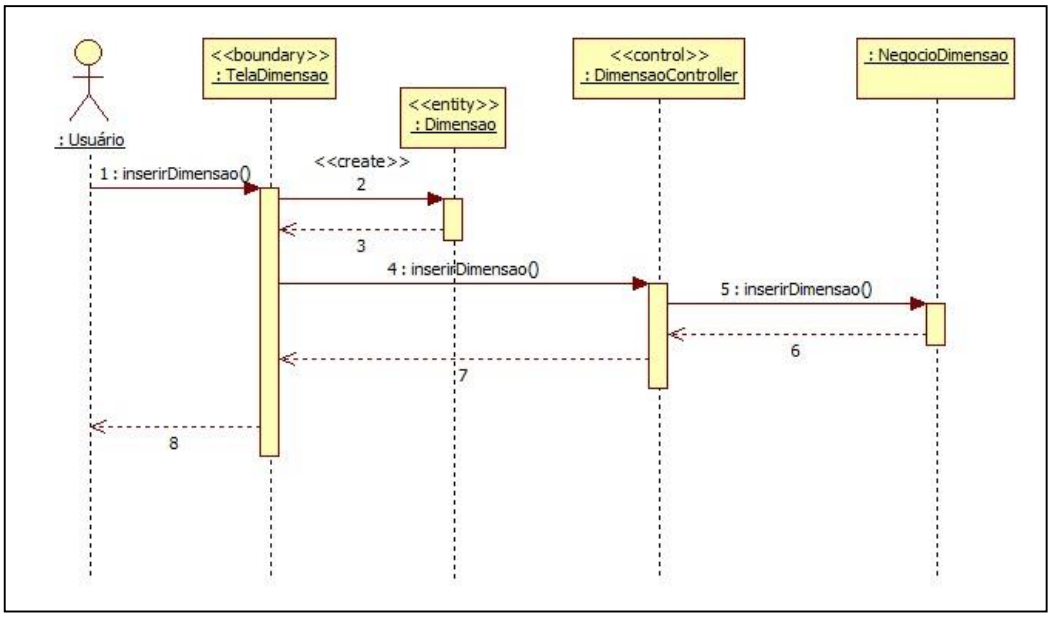

Figura 6: Diagrama de Sequência do caso de uso UC01-Dimensão.

3.2.5 Modelo Orientado a Objetos

Por fim, a modelagem de dados orientada a Objetos (OO) é apresentada na Figura 7. O mesmo foi concebido através de pesquisas anteriores que tiveram como um dos objetivos criar um conjunto de metadados que gerenciassem o comportamento de dados em um ambiente de DW, apresentado em (SANTOS; COSTA; NASCIMENTO, COLAÇO, 2012a) (SANTOS; NASCIMENTO, 2012b).

O modelo OO apresenta como principais entidades a dimensão, tabela auxiliar e o procedimento que irá conter ambas as entidades. As demais entidades referem-se ao relacionamento entre os atributos da dimensão e tabela auxiliar e suas restrições e comportamentos.

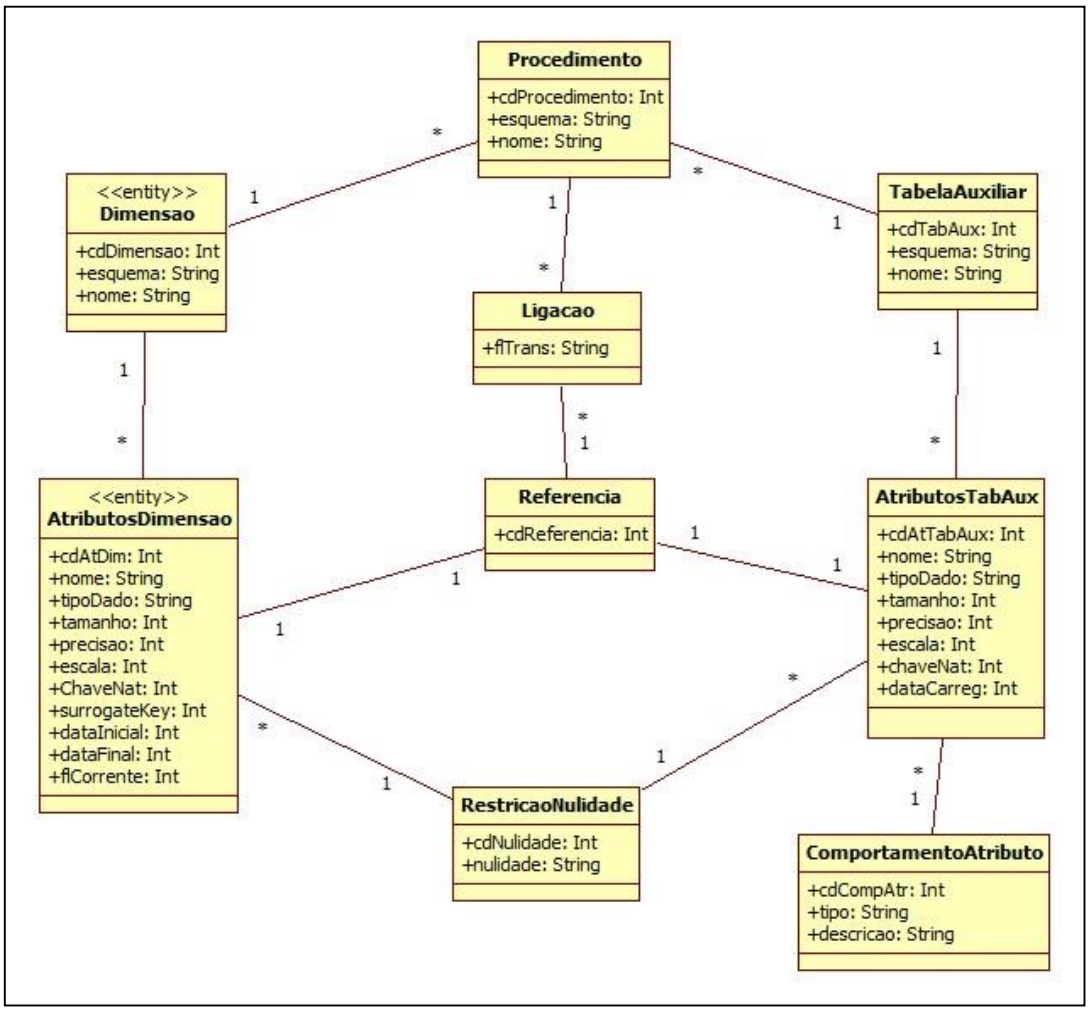

Figura 7: Modelo OO do *framework*.

3.3 Interfaces do *framework*

Posterior à etapa de design do projeto foi desenvolvido o *framework* FTUnit. O mesmo se deu de forma iterativa e incremental utilizando a linguagem C#. As próximas seções irão mostrar as principais *interfaces* do FTUnit e seu comportamento.

A Figura 8 apresenta a tela inicial da FTUnit. Em (1), acontece o gerenciamento das tabelas de dimensão, onde são configurados os metadados que compõem uma dimensão e seus tratamentos para histórico de dados. Em (2), pode ser realizado o gerenciamento das tabelas auxiliares (tabelas intermediárias que recebem dados dos sistemas transacionais), ou seja são configurados o nome e esquema da tabela, assim como os atributos que a compõe, suas características e comportamentos. No gerenciamento do procedimento, localizado em (3), deve ser informada a tabela auxiliar e a dimensão que irá compor o procedimento, além de criar o relacionamento entre os atributos dessas tabelas. Para finalizar, em (4), o usuário executa os testes unitários para o procedimento definido em (3).

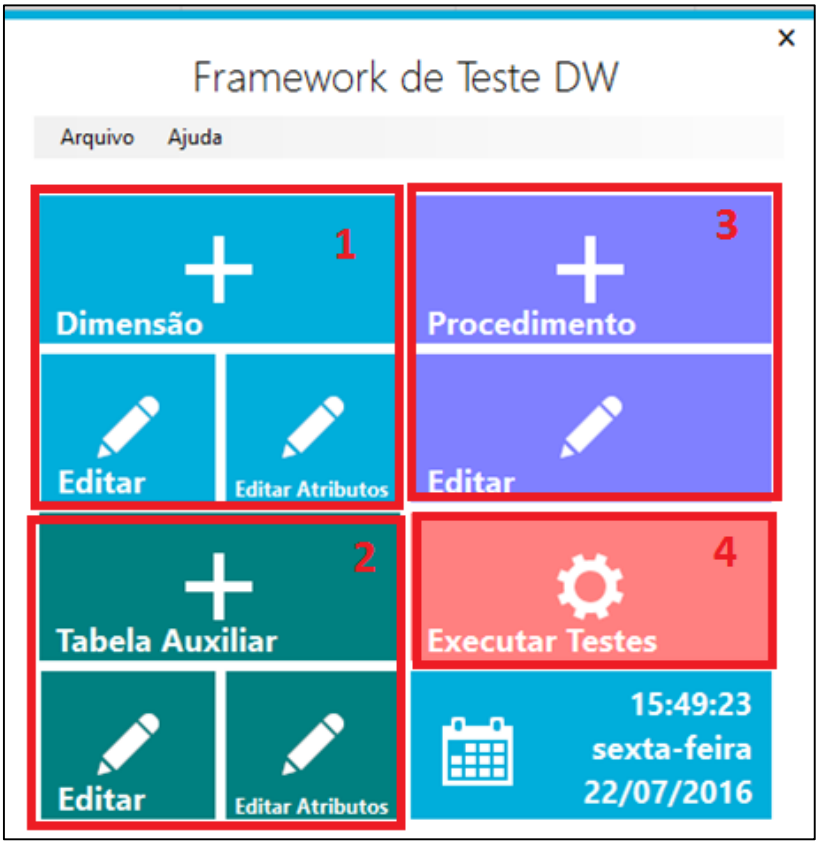

Figura 8: Página Inicial do FTUnit.

3.3.1 Dimensão

Antes de executar os testes sobre o procedimento, é necessário que seja configurado todo o ambiente de DW. O usuário ao selecionar a opção 1 da Figura 8 acessa a tela de configuração do esquema e nome da dimensão. Posteriormente, a tela (Figura 9) com o comportamento dos atributos da dimensão pode ser configurada. Nesse momento são inseridas as características de chave natural; *surrogate key*; restrição de nulidade; data inicial; data final; e, *flag* corrente.

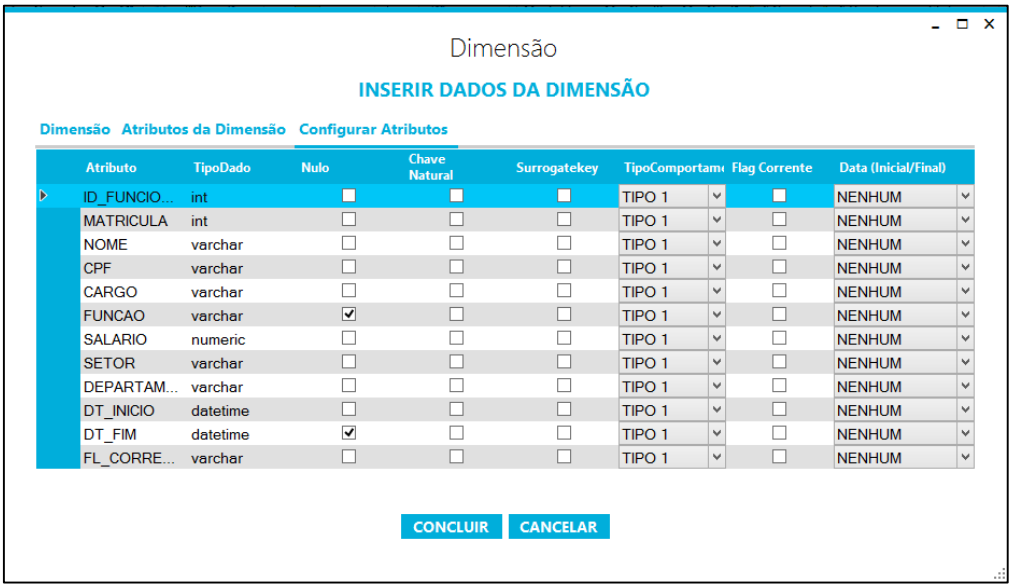

Figura 9: Tela de configuração de comportamento dos atributos de uma dimensão de funcionário.

3.3.2 Tabela Auxiliar

Bem semelhante à configuração da dimensão, na opção 2 da tela inicial (Figura 8) o usuário configura seu ambiente de *staging* (esquema e nome da tabela auxiliar). Posteriormente na Figura 10 são adicionados seus atributos e particularidades de chave natural, data de carga, e se estes recebem valor nulo ou não.

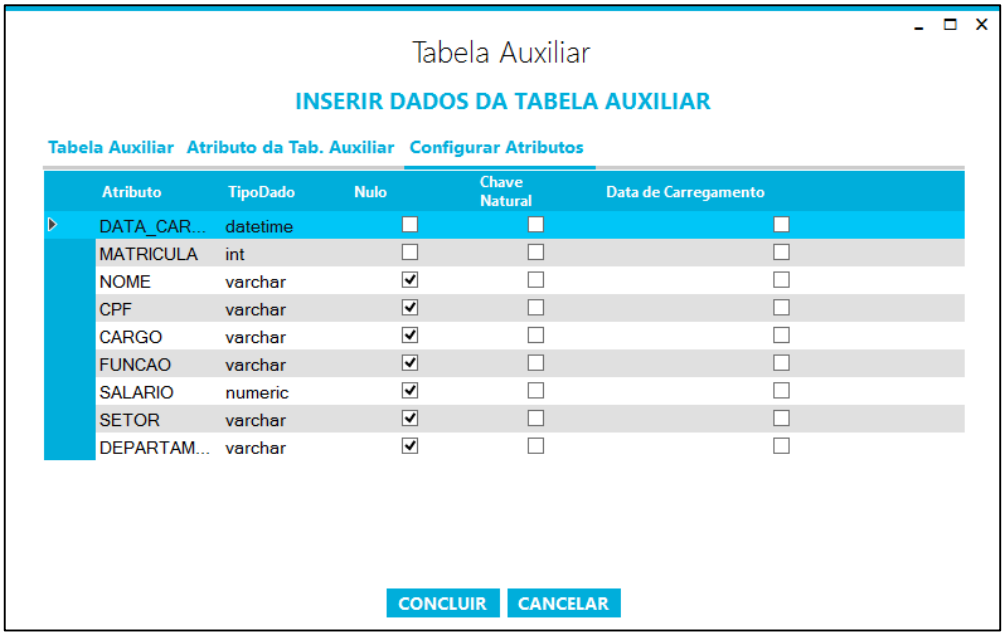

Figura 10: São listados os atributos da Tabela Auxiliar.

3.3.3 Procedimento

A última etapa, anterior à de execução de testes, ocorre com a configuração do procedimento (Figura 11 (A)), no qual é possível determinar o nome e esquema, além das tabelas que compõem o procedimento. Em (B) da mesma figura o usuário configura o relacionamento dos atributos da tabela auxiliar e da dimensão. O resultado do relacionamento entre o atributo da tabela auxiliar com o atributo da dimensão pode ser visto em (C).

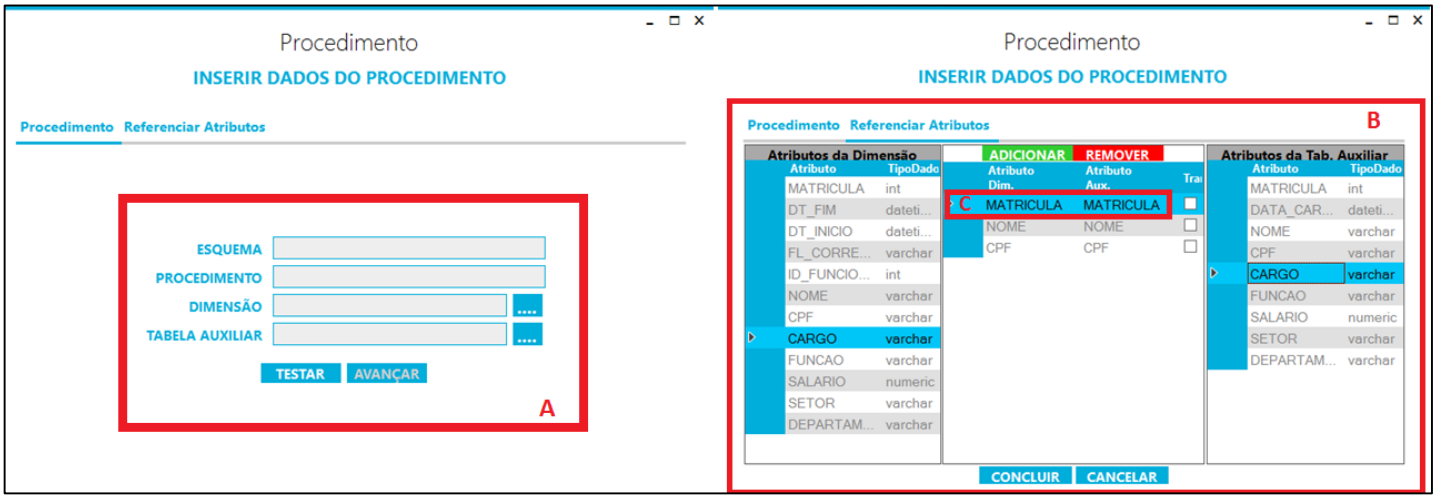

Figura 11: Tela de configuração do procedimento (A) e relacionamento dos atributos da tabela auxiliar e dimensão (B).

3.3.4 Executar Testes

A Figura 12 representa a tela onde é selecionado o procedimento, configurado anteriormente em (3) da Figura 8, e os Casos de Testes que o usuário deseja realizar sobre esse procedimento. O resultado da execução é apresentado na Figura 13, o qual contém os possíveis erros e a descrição do mesmo, caso haja algum problema no processo de carga de dados para o DW.

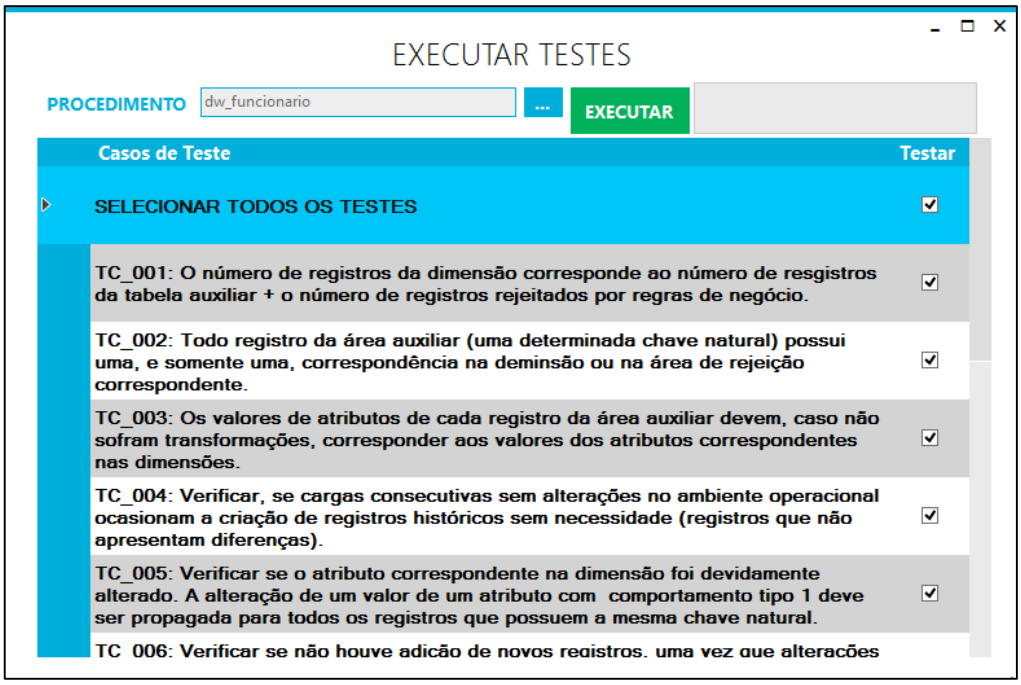

Figura 12: Tela de execução dos testes.

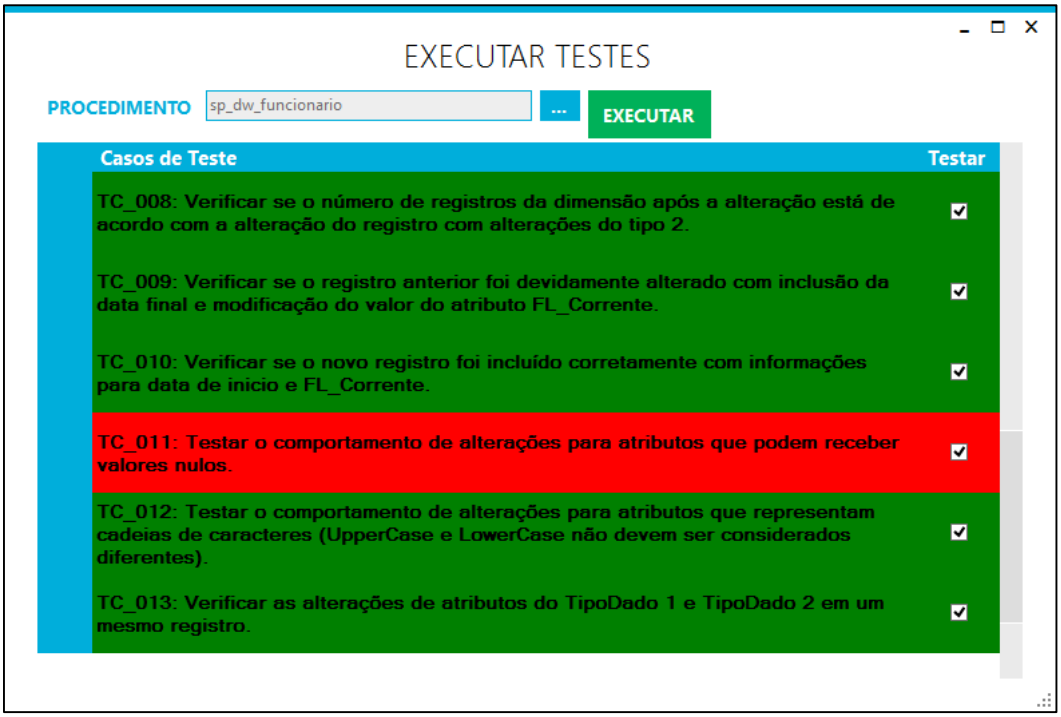

Figura 13: Tela com o resultado da execução dos testes.

Ao final da execução dos testes é gerado um relatório que pode ser visto na Figura 14. Esse relatório é um log de execução de cada teste unitário selecionado, informando passo a passo a execução dos mesmos, bem como os valores utilizados nos testes.

```
        CASO DE TESTE 001

DELETANDO DADOS DA TABELA AUXILIAR E DA DIMENSÃO

INSERINDO DADOS NA TABELA AUXILIAR

DADOS DA TABELA AUXILIAR - TB_AUX_FUNCIONARIO
-------------------------------------------------------------------------------------------
|DATA_CARGA  |MATRICULA  |NOME  |CPF  |CARGO  |FUNCAO  |SALARIO  |SETOR  |DEPARTAMENTO  |
-------------------------------------------------------------------------------------------
|2012-02-01 00:00:00.0|         43|   sol|  pro|    bus|     bus|     4.00|    pro|          com|
|2012-02-01 00:00:00.0|         17|   def|  def|    sol|     def|    47.00|    com|          pro|
|2012-02-01 00:00:00.0|          4|   bus|  bus|    def|     sol|    22.00|    bus|          sol|
|2012-02-01 00:00:00.0|         12|   com|  sol|    pro|     com|    33.00|    def|          bus|
|2012-02-01 00:00:00.0|         46|   pro|  com|    com|     pro|    38.00|    sol|          def|


EXECUTANDO PROCEDIMENTO - SP_DW_FUNCIONARIO

DADOS DA DIMENSÃO - DIM_FUNCIONARIO
-----------------------------------------------------------------------------------------------------------------------------------------
|ID_FUNCIONARIO  |MATRICULA  |NOME  |CPF  |CARGO  |FUNCAO  |SALARIO  |SETOR  |DEPARTAMENTO  |DT_INICIO  |DT_FIM  |FL_CORRENTE  |
-----------------------------------------------------------------------------------------------------------------------------------------
|           11838|         43|   sol|  pro|    bus|     bus|     4.00|    pro|          com|2012-02-01 00:00:00.0|    null|            S|
|           11839|         17|   def|  def|    sol|     def|    47.00|    com|          pro|2012-02-01 00:00:00.0|    null|            S|
|           11840|          4|   bus|  bus|    def|     sol|    22.00|    bus|          sol|2012-02-01 00:00:00.0|    null|            S|
|           11841|         12|   com|  sol|    pro|     com|    33.00|    def|          bus|2012-02-01 00:00:00.0|    null|            S|
|           11842|         46|   pro|  com|    com|     pro|    38.00|    sol|          def|2012-02-01 00:00:00.0|    null|            S|


RESULTADO DO TESTE: OK
```

Figura 14: Arquivo de log de cada Execução de Testes.

4 EXPERIMENTO I

Experimentation in the Industry for Automation of Unit Testing in a Business Intelligence Environment

Igor Peterson Oliveira Santos[a], André Vinícius R. P. Nascimento[b], Juli Kelle Góis Costa[a], Methanias Colaço Júnior [a,b], Wenderson Campos Pereira[b]

[a] Postgraduate Program in Computer Science - PROCC.
UFS – Federal University of Sergipe
São Cristóvão/SE - Brasil.
{igorp.ita, julikelle}@hotmail.com

[b] Competitive Intelligence Research and Practice Group – NUPIC
Information Systems Departament - DSI
UFS – Federal University of Sergipe
Itabaiana/SE - Brasil.
andreviniciusnascimento@gmail.com, {mjrse, wenderson_se}@hotmail.com

***Abstract*— This paper presents an approach to automate the selection and execution of previously identified test cases for loading procedures in Business Intelligence (BI) environments based on Data Warehouse (DW). To verify and validate the approach, a unit test *framework* was developed. The overall goal is achieve efficiency improvement. The specific aim is reduce test effort and, consequently, promote test activities in data warehousing process. A controlled experiment evaluation was carried out to investigate the adequacy of the proposed method for data warehouse procedures development. The results of the experiment show that our approach clearly reduces test effort when compared with manual execution of test cases.**

Keywords*— Business Intelligence; Data Warehouse; Software Testing; Data Quality; Experimental Software Engineering.

1. INTRODUCTION

Information represents a crucial factor for companies in improving processes and decision making. To assist the strategic areas of the organizations business intelligence (BI) environments are presented as sets of technologies that support the analysis of data and key performance indicators [1].

A central component of BI systems is a Data Warehouse (DW), a central repository of historical data. The idea behind this approach is to select, integrate and organize data from the operational systems and external sources, so they can be accessed more efficiently and represent a single view of enterprise data [1, 2, 3].

Despite the potential benefits of a DW, data quality issues prevent users from realizing the benefits of a business intelligence environment. Problems related to data quality can arise in any stage of the ETL (Extract, Transform and Load) process, especially in the loading phase. The main causes that contribute to poor data quality in data warehousing are identified in [4].

The lack of availability of automated unit testing facility in ETL tools is also appointed as cause for the poor data quality [4]. The low adoption of testing activities in DW environment is credited to the differences between the architecture of this environment and architectures of the generic software systems. These differences mean that the testing techniques used by the latter need to be adjusted for a DW environment [5, 6].

The test of ETL procedures is considered the most critical and complex test phase in DW environment because it directly affects data quality [7]. ETL procedures, more precisely the loading routines, exhibit the same behavior as database applications. They operate on initial database state and generate a final consistent database state. So, a black-box approach, which combines the unit and application behavior of loading procedures, is proposed. In this approach, the concern is with the application interface, and not with the internal behavior and program structure [8,9,10]. This approach to ETL routines, in some environments, may be the only option, since the use of ETL tools in DW environment produces codes or packages whose internal structure is not known.

This paper presents the result of the construction and experimentation of a unit testing *framework* to improve efficiency of loading procedures in a BI environment. Using a black-box approach and treating the loading routines under the application point of view, the *framework* generates initial and final states of the database in function of test cases previously specified. The expected behavior of the routines, the selection of test cases and database conditions used are determined by procedures metadata.

Thus, considering an industrial environment, this work aims address the following research question: "*A customized unit testing framework can increase the productivity of developers during the testing process in a DW?*" To answer, our experimental

evaluation analyzed a real context. The results showed numbers that indicate test effort reduction.

The remainder of this paper is structured as follows. Section 2 presents the test *framework*. Section 3 describes the experiment definition and planning. Section 4 presents the operation of the experiment. Section 5 reports the results of the experiment. In Section 6, related works are presented. Finally, section 7 contains the conclusion and future works.

2. FRAMEWORK

A. Architecture

The *framework* will be used to perform tests under the black-box approach. The code and the internal structure of the routines will not be examined. The test cases previously implemented, will be selected according to the characteristics of the routine being tested. Each routine, in order to be covered by the *framework*, must have a set of metadata registered. This set of metadata was defined from the schema presented in [11].

B. FTUnit tool

The FTUnit tool is a *framework* used to perform unit tests in loading procedures of a DW environment. It has been developed in C#. It is available for download at <http://ftunit.wordpress.com/>.

Figure 1 illustrates the main part of the initial screen of FTUnit. In (1), occurs the management of the dimension tables, where metadata, which make up a dimension and its treatments to historical data, are configured. In (2) can be performed management of auxiliary tables (intermediary tables which receive data from the transactional systems), in other words, name and table layout are configured, as well as the attributes that compose it, its characteristics and behaviors. In management procedure, located in (3), must be informed the auxiliary table and the dimension that will compose the procedure, as well as creating the relationship between the attributes of these tables. Finally, in (4), the user runs the unit tests for the procedure defined in (3).

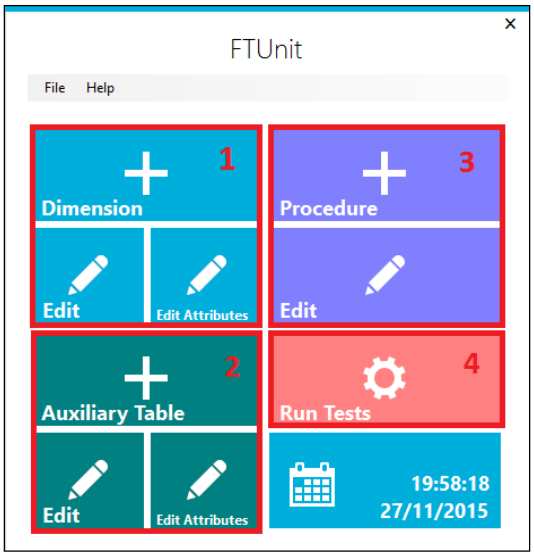

Figure 1: FTUnit Home Screen.

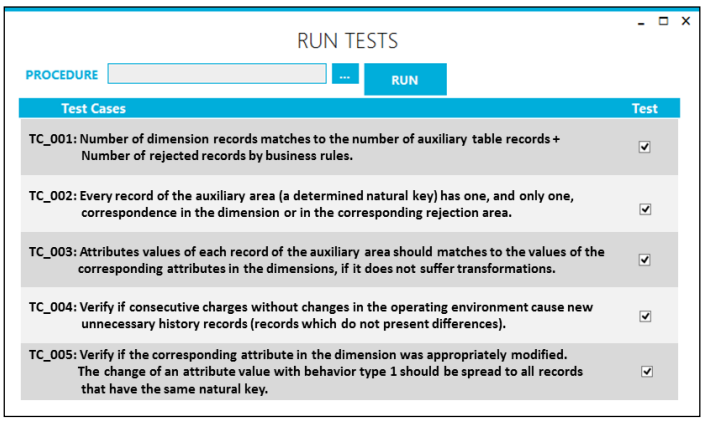

Figure 2: Test execution screen.

Figure 2 shows the selection of the procedure the user wants to run test cases. Once the procedure is selected, the *framework*, based on procedures metadata which defines, among other things, historical behavior of attributes, displays all the tests cases that apply. From this point the user can choose, based on some criteria or development stage, only the test cases that should be executed by the tool, as follows:

- First test load to dimension attributes with behavior type 1[4].
- Second test load to dimension attributes with behavior type 1.
- Unit testing to dimension attributes with behavior type 1.
- Integrity test for dimension attributes with behavior type 1.

[4] There is no storage of historical data [16].

- First test load to dimension attributes with behavior type 2[5].
- Data modified test to dimension attributes with behavior type 1.
- Data modified test to dimension attributes with behavior type 2.
- Data modified test to dimension attributes with behavior type 2. Changes to more than one behavior type 2 for dimension attribute.
- Data modified test to dimension attributes with behavior type 2. Changes in attributes that can have initial null values.
- Data modified test to dimension attributes with type 1 in dimensions with attributes of type 2 with stored history.

At the end of the execution, the *framework* generates an execution log report of each selected test case, informing step by step its implementation and the values used in the tests.

3. EXPERIMENTAL DEFINITION AND PLANNING

Our work is presented here as an experimental process. It follows the guidelines by Wohlin et al. in [12]. In this section, we start introducing the experiment definition and planning. The following sections, will direct to the experiment execution and data analysis.

A. Goal definition

Our main goal is to evaluate the use of automated test execution to loading routines in a Data Warehouse environment.

The experiment will target developers of ETL processes for BI environments with at least 2 years of experience in the market and one year of experience in ETL programming. The goal was formalized using the GQM model proposed by Basili and Weiss [13]:

- **Analyze** the use of a DW unit testing *framework*
- **With the purpose of** evaluate (against manual testing)
- **With respect to** the efficiency of the process of executing test cases
- **from the point of view of** developers and decision support managers
- **in the context of** programmers in a BI company.

---

[5] There are treatment and storage of the historical data, by creating new records [16].

B. Planning

1) Hypothesis Formulation

The research question for the experiment that needs to be answered is this: "*A customized unit testing framework can increase the productivity of developers during the testing process in a DW?*"

To evaluate this question, it will be used a measure: Average time for Manual testing and Automation Testing. Having the purpose and measures defined, it will be considered the hypothesis:

H_{0time}: the execution of automated and manual testing has same efficiency. ($\mu_{ManualTestingTime} = \mu_{AutomationTestingTime}$).

H_{1time}: the execution of automated testing is more efficient than the execution of manual testing. ($\mu_{ManualTestingTime} > \mu_{AutomationTestingTime}$).

Formally, the hypothesis we are trying to reject is $H_{0time.}$ To ascertain which of the hypotheses is rejected, will be considered the dependent and independent variables that are in Figure 3.

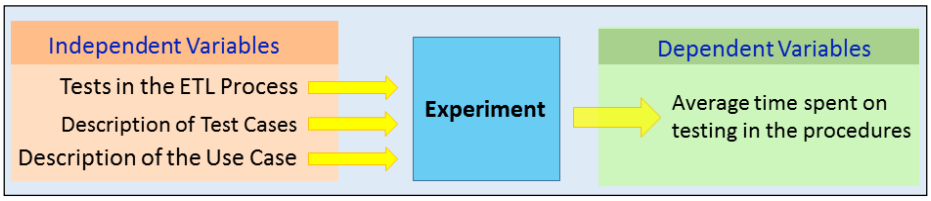

Figure 3: Dependent and Independent variables of the experiment.

a) Independent Variables

Next, the independent variables of the experiment are described.

Description of Test Cases Used in the Experiment: The loading routines for the DW environment are quite discussed in [1,2,14,15]. Alternative approaches to the loading of dimensions can be found in [16]. Algorithms for loading routines for the various types of dimensions can be found in [17,18]. Test Cases categories for ETL routines are pointed in [6,19]. This material, together with the extensive experience of the authors in DW projects in the public and private sectors, provided the basis for the elaboration of categories and test cases to be considered by the *framework*. The following categories are contemplated by the *framework*: a) Unit tests and relationship; b) Number of records between source and destination; c) Transformations between source and destination; d) Processing of incorrect or rejected data; e) Null values

processing; f) Behavior type 1, type 2 and type 3[6] for dimensions attributes; g) Hybrid approaches for the treatment of historical dimensions.

TABLE I. FEATURES OF THE DIMENSION DBO.DIM_EMPLOYEE AND ITS ATTRIBUTES REGARDING THE HISTORICAL STORAGE.

Dimension Name:	dbo.dim_employee	
Treatment historical Type:	Type 1 and 2	
Attribute	Historical Type	Attribute Paper
id_employee		**Surrogate Key**
cod_registration		**Primary Key**
name	**1**	
cpf	**1**	
title	**2**	
job	**2**	
salary	**2**	
sector	**2**	
department	**2**	
dt_initial		**Initial Date**
dt_end		**End Date**
fl_current		**Current Flag**

Description of the Use Case Used in the Experiment: the characteristics of the use case chosen for the validation study were based on practical situations reported by the selected programmers.

For the use case of the experiment, the goal was to generate procedure to perform loads of Staging Area, from employee table to the employee dimension. At this time, the dimension has an historical storage, Type 2 for some attributes. The other attributes are Type 1.

Table I shows the characteristics of the employee dimension in DW environment. For the Type 2 treatment in dimensions, new attributes are used for the historical storage. They are: the start date, which represents the date on which the record was recorded; the end date which is the date when the record is no longer current; and finally, the attribute that identifies whether it is or not a current record. These are respectively shown in Table I, with the names of: dt_initial; dt_end and fl_current.

Tests in the ETL Process: The ETL tests used in this work, have two types of treatments for performing the experiment: 1) Manual Testing: manual testing execution,

[6] There is treatment of historical data, by storing in various attributes of the same data record. So, can be created how many columns as desired for the dimension [16].

based on the test cases, in the ETL procedures, in SQL to load data defined in the use case already presented earlier; 2) Automation Testing: execution of tests based on test case, in the SQL code for the same use case, using the proposed tool in this work, FTUnit.

b) Dependent Variables

It were used a measure as a dependent variable: Average time, for manual testing and automation testing, measured using a stopwatch, considering the average time spent on testing in the procedures.

2) Participants Selection

The selection process of the participants will be done for convenience, making the type of sampling per share in which will be preserved the same characteristics of interest present in the population as a whole. The contributor to be chosen will be Qualycred (www.qualycred.com), a company that provides consulting in BI solutions for industry. This company will provide for the execution of the experiment, ten programmers with four years of experience in other areas and one year of experience working specifically with ETL for DW, in SGBD SQL SERVER.

3) Experiment Project

The experiment was projected in a paired context, in which a group will evaluate both approaches: Manual and Automated execution of test. For understanding the execution of the test to be done, ten test cases and one use case (seen in Independent Variables section) were elaborated, which will be presented in a well detailed way to the programmers.

The experiment will be separated into two groups of participants. Will be drawn 5 programmers to start the tests to the rules presented in the Employee Use Case, with the execution of manual testing and, shortly after, the execution of automation testing. The other participants in parallel, will make the tests made to rules presented in same use case, with the implementation of the automation testing and, shortly after, with the execution of the manual testing. Thus, the randomness will be enhanced, not prioritizing the manual or automated learning.

4) Instrumentation

The instrumentation process initially proceeded with the environment setup for the experiment and planning the data collect. It was conducted in a computer lab at Federal University of Sergipe - UFS. The Participants of the experiment had the same

working conditions. The computers were adjusted to possess same settings. Listed below are the technology, the installed tools and artifacts used.

SQL Server 2008. It served as a basis for the storage of the identified metadata, and consequently, has been used to store the metadata of FTUnit, described in section 2.

Framework of Unit Testing in Sql Code (FTUnit). The FTUnit was described section 2 of this paper. The version to be used in the experiment, due to the participants, runs Unit Testing Cases for charging procedures in SQL code, T-SQL language (Transact-SQL), involving behaviors of Types 1, 2, and 3 for the treatment of historical dimensions.

Environment Created and Produced Artifacts. Some of the tables that make up the DW environment used in the experiments were the following: a) auxiliary employee table; b) dimension of employee with attributes types 1 and 2. These are described below.

Figure 4 contains the representation of a load data from TB_Aux_Employee (auxiliary table of employees) to the DIM_Employee (dimension of employees) that represents a dimension of types 1 and 2. To this dimension, the attributes that match the type 1 are: name and CPF. The attributes of type 2 are: title, job, salary, sector and department.

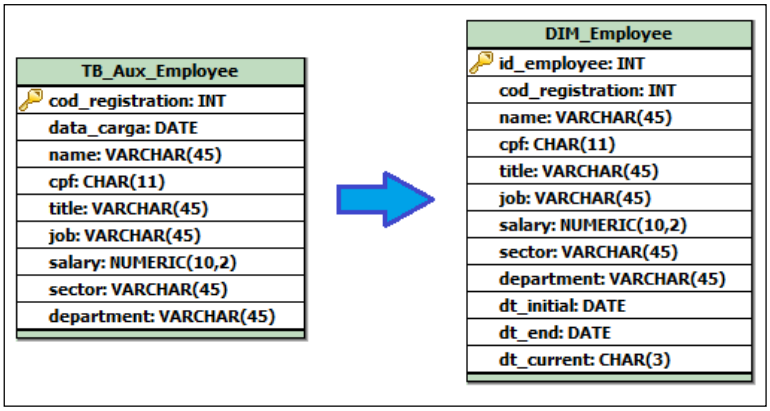

Figure 4: Charge for a dimension of employees with behaviors of types 1 and 2.

4. EXPERIMENT OPERATION

A. The preparation

The following are listed the preparation steps for the execution of the experiment.

1) DW environment Creation: *in this phase was defined and created the DW environment with the dimensional schemes and staging area. These artifacts served as the basis for the entire experiment.*

2) Definition of Test Cases for loading routines: *were defined test cases to be followed by the developers of the experiment. These test cases were applied to a loading routine previously created.*

3) Review of basic concepts of loading routines for the programmers: *a review of the loading routines, for DW environments with the selected developers, was performed.*

4) Training in testing *framework*: *a training with the programmers was realized to become familiar with the tool.*

In short, all computers were prepared with the same settings, so programmers were on the same working conditions. Moreover, it was presented to each programmer, a printed document containing a detailed description of Use Case and test cases that would be used by them, in case of any doubts.

B. Execution

At the end of the previous steps, the experiment was initiated, it occurred according to the plan described in section 3.

The evaluation of the tool at the end of the experiment, made by the professionals, was positive, since they have commented that the use of the tool have contributed to the reduction of time in the test procedures.

1) Data Collection

It was calculated the time spent by each developer for both manual and automated tests, of all test cases for the Employee Use Case, taking into account the time for testing and all necessary settings in FTUnit. Under supervision, each programmer reported the completion and was recorded the time on a timer, used for this purpose. The result of these collected data will be presented in section 5 of this paper.

C. Data Validation

In order to perform the experiment, one factor was considered, Test of the ETL Process, and two treatments, manual and automated tests, using the FTUnit tool. Facing this context, the average of testing time was computed.

As an aid to analysis, interpretation and validation, we used two types of statistical tests, Shapiro-Wilk Test and the T Test. Shapiro-Wilk test was used to verify normality of the samples. The T test was used to compare the average of the two paired samples [12]. All statistical tests were performed using the SPSS tool [20].

5. RESULTS

A. Analysis and Data Interpretation

To answer the question of research, the following dependent variable was analyzed: The time to the testing process of each procedure.

1) Time spent in the testing process.

Table II displays the results related to the testing time by each participant for the Employee Use Case. The results show that the average time of the developers for manual testing was 54.4 minutes, and 20.5 minutes for the automatic one.

These results suggest that the automated testing procedures have, on average, shorter testing time, as compared with the same test procedure performed manually by programmers with experience in the area. Thus, from this preliminary analysis of the data, it is assumed that the answer to the Research Question would be "yes". The execution of automated testing can increase the productivity of developers during the testing process in a DW, since automation testing obtained a difference of approximately 35 minutes. But is not possible to make such a claim without sufficiently conclusive statistical evidence.

Thus, first, we established an apriority significance level of 0.05. The Shapiro-Wilk test ensured that the sample was normally distributed. As seen in Table III, we found p-values of 0.659 and 0.311 for execution of manual and automated testing, respectively. As the p-value is the lowest possible significance with which it is possible to reject the null hypothesis, and they are larger than 0.05, we cannot reject the hypothesis that the data is normally distributed.

Finally, as the samples are not independent, the hypothesis test applied in this context was the T-Test, characterized as parametric for paired samples, which only requires normality of the samples. In Table IV, we obtained the p-value of 0.000. This means the p-value found is less than 0.0001, so we have more than 99% certainty for the valued context. Thus, it was confirmed the evidence of a difference between the averages of 33.9. As the significance test is lower than 0.05, it is possible to reject the null hypothesis. Consequently, we cannot reject the alternative hypothesis that the execution of automated testing is more efficient than the execution of manual testing.

Table II. Average Time (in minutes) to Execute The Tests of the use cases.

	Employee Use Case	
	Manual (minutes)	**Automatic (minutes)**
Programmer 1	44	20
Programmer 2	42	27

Programmer 3	52	21
Programmer 4	45	23
Programmer 5	72	8
Programmer 6	28	19
Programmer 7	63	12
Programmer 8	87	25
Programmer 9	40	22
Programmer 10	71	28

Table III. Shapiro-Wilk Test. (SOURCE: SPSS TOOL – IBM [20]).

	Statistic	**df**	**Sig.**
Manual Testing	,949	10	,659
Automated Testing	,914	10	,311

Table IV. T-Test Versus Time of the Tests. (Source: SPSS Tool – IBM [20]).

		Paired Differences							
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference		t	df	Sig(2-tailed)
					Lower	Upper			
Pair 1	ManualTesting-AutomatedTesting	33,90	19,85755	6,27951	19,69477	48,10523	5,399	9	,00

B. Threats to validity

In spite of having achieved statistical significance in the study, the following threats to the validity must be considered.

Threats to internal validity: Although participants have been trained to use the tool, they do not use it daily. This lack of constant contact with it may have affected the results, which could be even better, pro-tool. The tool training was conducted at the beginning of the experiment, considering a phenomenon studied by psychology called *Demand Characterization* - which considers that an experimental artifact may have an

interpretation of the purpose of the experiment by the participants. This can lead to change of unconscious behavior, to adapt to this interpretation [21]. According to this concept, this training could be harmed the progress of the experiment, but to mitigate this factor, can be said that had been used at least two different approaches*: The More The Merrier* and *Unobtrusive Manipulations and Measures* [21]. Respectively, the first, to avoid bias with a single experimenter, the experiment had another researcher to conduct the experiment and an instructor for the tool, not involved with the research. The second guided us not to say which factors and metrics would be assessed, so that the participants had no clues about the research hypothesis.

Threats to external validity: The low number of participants can be a threat, since it can negatively influence the results of the experiment. This threat was mitigated with the convenience of the selection of programmers skilled in the ETL area for BI environment.

Threats to the construction validity: The Specifications for the use case and test cases may not have been very clear to the understanding of some programmers. This threat was mitigated with the prior reading and analysis of the understanding, made by 3 ETL developers.

6. RELATED WORK

Through literature reviews, with systematic approaches, were not found strongly related work for automated unit tests in ETL tools. Consequently, the absence of ETL tools with these characteristics may contribute to a lower integrity and a lower quality of data, essential in large banks of decision support data.

Some moderately related works also seek solutions for the automatic execution of Test Cases in DW environments. In [6] it is presented a directed models approach for automatic generation and execution of test cases based in formal models of systems. The formal model adopted is based on the UML language. The approach also depends on creating an extension of UML language that can capture the transformations used in a *Data Warehousing* process.

The QuerySurge [22] tool, developed by RTTS Company, presents the possibility of automatic execution of unit tests. This approach differs from ours, since the goal is to work with programmed unit tests, not pre-defined by the tool. The approach adopted is to create scripts that can capture operational environment data and dimensional schema

for comparison. The tool does not use metadata to work with already known transformations, as it happens in our approach.

Once the *framework* generates test cases based on characteristics of the loads procedures being implemented, it can be extended and used to test load routines created for any ETL tools. So far was not found in literature any similar approach, so we could make a comparison. The more similar tool to the proposed work is the *framework* [23]. However, this one represents a generic *framework* for database applications and has no particularity regarding to loading routines for a Data Warehouse environment.

7. CONCLUSIONS

Business Intelligence requires valid, consistent, and complete organizational data. These quality items represent constant concerns for companies in the process of use of decision support systems.

In this paper, we presented the proposal of using a unit testing *framework* for loading routines in a BI environment based on Data Warehouse. The motivation for adopting this approach meets the problems pointed out in [24], as the main causes for the poor quality of data in a DW environment. Another motivation, also pointed in [6,7,8] is the need to adopt different strategies, considering the differences between traditional environments and DW environments, which can contribute to the adoption of testing processes.

In this context, this work presents important contributions to increasing the productivity and quality in software engineering for loading routines of DWs, and encourages experimentation in an industrial environment. The *framework* encapsulates a method to accelerate and improve the quality of ETL process tests based on SQL. It is noteworthy that the safe and efficient execution of procedures in SQL directly in the database is an option considered by much of the industry, requiring tools to support tests in this type of approach in software engineering.

The proposed *framework* presents test cases previously defined which cover the main categories of tests applied to loading routines. Through a set of metadata that defines the characteristics of the routines, the *framework* selects test cases to be applied, generates the initial states of the database, executes the routines, performs test cases, analyzes the final state of the database and generates a report with the errors encountered during the execution of each test case.

By virtue of what we have seen above and the *framework* innovation, the presentation of this experiment will support the adoption of the same or the creation of a similar approach for companies that use this type of strategy.

As future work, experiments will be done evaluating the use of the proposed *framework* against a generic database application test *framework*, the DBUnit [23] which had been constructed specifically for database application tests.

REFERENCES

[1] Colaço Jr., M.: Projetando sistemas de apoio à decisão baseados em Data Warehouse. 1st ed., Rio de Janeiro: Axcel Books (2004)

[2] Kimball, R., Ross, R. M. and Thomthwaite, W.: The Data Warehouse lifecycle toolkit. 2nd. ed., Indianapolis, Indiana: Wiley Publishing Inc (2008)

[3] Inmon, W. H.: Building the Data Warehouse. 4th ed., Indianapolis, Indiana: Wiley Publishing Inc (2005)

[4] Ranjit S. and Kawaljeet, S.: A Descriptive Classification of Causes of Data Quality Problems in Data Warehousing. 7 v. IJCSI International Journal Of Computer Science Issues (2010)

[5] Deshpande, K.: Model Based Testing of Data Warehouse. IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 2, No 3 (2013)

[6] Elgamal, N., , Elbastawissy, A. and Galol-edeen, G.: Data Warehouse Testing. EDBT/ICDT '13, Genoa, Italy (2013)

[7] Golfarelli, M. and Rizzi, S. A.: Comprehensive Approach to Data Warehouse Testing. ACM 12th International Workshop on Data Warehousing and OLAP (DOLAP '09), Hong Kong, China (2009)

[8] Myers, G. J., Badgett, T. and Sandler, C.: The Art Of Software Testing. 3rd ed., New Jersey: Wiley (2012)

[9] Pressman, R. S.: Engenharia de software: Uma abordagem profissional. 7th ed., São Paulo: AMGH Editora Ltda (2011)

[10] Sommerville, I.: Engenharia de Software. 9th ed., São Paulo: Pearson (2011)

[11] Hidden reference to not identify authors, it will be replaced in the final version.

[12] Wohlin, C., et al.: Experimentation in Software Engineering: An introduction. USA: Kluwer Academic Publishers (2000)

[13] Basili, V. and Weiss, D.: A Methodology for Collecting Valid Software Engineering Data. In: IEEE Transactions On Software Engineering, v.10 (3): 728-738, November (1984)

[14] Kimball, R. and Ross, M.: The Data Warehouse toolkit: The complete Guide to Dimensional Modeling. 2nd ed., John Wiley and Sons, Inc (2002)

[15] Kimball, R.: The Data Warehouse ETL Toolkit. 1st ed., Wiley India (P) Ltd (2004)

[16] Santos, V. and Belo, O.: No Need to Type Slowly Changing Dimensions. IADIS International Conference Information Systems (2011)

[17] Hidden reference to not identify authors, it will be replaced in the final version.

[18] Hidden reference to not identify authors, it will be replaced in the final version.

[19] Cooper, R. and Arbuckle, S.: How to thoroughly test a Data Warehouse. Proceedings of STAREAST, Orlando (2002)

[20] SPSS, IBM Software, http://goo.gl/eXfcT3

[21] Orne, M. T.: Sobre a psicologia social da experiência psicológica: Com referência particular para exigir características e suas implicações. (1962)

[22] QuerySurge, RTTS, http://www.querysurge.com/

[23] DBUnit, http://dbunit.sourceforge.net/

[24] Singh, R. and Singh, K.: A Descriptive Classification of Causes of Data Quality Problems in Data Warehouse. IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 3, No 2, May (2010)

CAPÍTULO 5

5 EXTENSÃO DO EXPERIMENTO I

Este capítulo apresenta-se como uma extensão do primeiro experimento, exposto no capítulo anterior. Neste foi apresentado a execução de um experimento controlado para a automação de execução de casos de testes de acordo com a abordagem manual. O objetivo é responder a seguinte pergunta: “a execução de testes automáticos pode aumentar a produtividade dos programadores durante o processo de teste em um DW?”. Os resultados foram também apresentados e discutidos no mesmo capítulo.

O mesmo experimento contemplou também como objetivo verificar e analisar o número de erros de codificação dos casos de testes usando a abordagem manual e automática (*framework* de Testes). As próximas seções informam as hipóteses definidas para o experimento, seus resultados e análises.

5.1 Planejamento

5.1.1 Formulação de Hipóteses

A questão de pesquisa para o experimento que precisa ser respondidas aqui neste capítulo é a seguinte: a execução de testes automáticos pode reduzir ou eliminar erros nas rotinas de carga em um DW?

Para avaliar estas questões, será utilizada a métrica como variável dependente: Média de grau de criticidade de erros de codificação, calculada considerando: (a) Registros carregados incorretamente para o DW: (1) Erro e falta de tratamento histórico para atributos das dimensões; (2) Registros duplicados; (3) Não atualização dos dados.

(b) Utilização de matriz GUT (sigla para Gravidade, Urgência e Tendência) (MARSHALL et al., 2011) para gerenciamento de problemas através dos erros citados no item (a), bem como cálculo do grau de criticidade.

Tendo o objetivo e a métrica definidos, será considerada a hipótese:

HIPÓTESE

- H_{0erros}: A execução dos testes automática e manual tem a mesma eficácia.

($\mu_{grauErrosCodificaçãoTesteManual} = \mu_{grauErrosCodificaçãoTesteAutomático}$).

- H_{1erros}: A execução de testes automática é mais eficaz que a execução de testes manual.

($\mu_{grauErrosCodificaçãoTesteManual} > \mu_{grauErrosCodificaçãoAutomática}$).

Para a hipótese, a $H_{0,}$ é a hipótese que se deseja rejeitar. A hipótese alternativa, $H_{1,}$ a que se deseja não rejeitar. Para averiguar quais das hipóteses são rejeitadas, serão consideradas as variáveis dependentes e independentes apresentadas no capítulo anterior.

5.2 Resultados

Após a operação do experimento realizado no capítulo anterior, os resultados obtidos para resolver a métrica discutida neste capítulo é a apresentada nesta seção.

Como auxílio para análise, interpretação e validação, foi utilizado como teste estatístico o Teste de Wilcoxon. Este compara as médias das amostras pareadas que não obtiveram normalidade nos dados, verificando a magnitude da diferença (WOHLIN et al., 2000). O teste estatístico foi feito utilizando a Ferramenta SPSS – IBM (SPSS, 2014).

5.2.1 Análise e Interpretação de Dados

Para responder à questão, foi analisada a seguinte variável dependente: *Número de erros de codificação*– número de erros de codificação dos casos de testes encontrados usando a abordagem manual e automática.

Erros de Codificação

Para responder à Questão de Pesquisa, foram comparados os erros, contabilizados levando em conta o esforço através da matriz GUT (Gravidade, Urgência e Tendência) (MARSHALL et al., 2011) (Quadro 30), encontrados após execução dos casos de testes de forma manual e automática, para o Caso de Uso apresentado no capítulo anterior. Os resultados relacionados ao grau de criticidade de erros de codificação por cada participante são apresentados na Tabela 1. Estes erros foram encontrados com abordagem *blind,* com o auxílio de um usuário de negócio com conhecimento do sistema OLTP, não tinha conhecimento do experimento.

Na matriz GUT, Quadro 30, são definidos os problemas que foram encontrados nos procedimentos gerados no experimento. Além disso, é estabelecido o valor para os índices de gravidade, urgência e tendência de erros para carga de dados em um *DW*.

O primeiro problema detectado (Quadro 30) foi a eliminação de antigos registros e inserção dos mesmos para os dados mais atuais provenientes da área de *Staging*, para atributos do tipo 1 (sem necessidade de guardar histórico). Esse acontecimento se torna um problema por violar os valores das *Surrogate Keys* das dimensões, visto que, se os registros já estiverem vinculados à tabela de fatos (indicadores de desempenho do negócio), estes não conseguirão relacionar-se com os registros da dimensão, por haver alterações nos valores das chaves estrangeiras.

Quadro 30: Matriz GUT.

Problemas		**G**	**U**	**T**	**Grau Crítico**
1	Eliminação e inserção de registros (com comportamento tipo 1) já encontrados na dimensão.	3	3	3	27
2	Falta de atualização para alguns atributos com comportamento do tipo 1.	3	3	4	36
3	Erros de valores nulos para atributos com comportamento tipo 2.	3	3	3	27
4	Duplicação de dados.	4	4	4	64
5	Não atualização de dados.	5	5	5	125
6	Erro no tratamento histórico para atributos tipo 2.	5	5	5	125

No segundo caso, linha 2 conforme Quadro 30, ocorre a falha na atualização para atributos com comportamento do tipo 1. O problema 3 está relacionado com os atributos

do tipo 2 que não apresentam modificações corretas para os valores nulos. A duplicação de dados, problema 4, está atrelado à inconsistência e redundância em um ambiente de DW. Esta redundância leva a altos custos de armazenamento e acesso aos dados.

Os piores casos envolvem os problemas 5 e 6. A não atualização de dados (problema 5) com o tratamento correto de histórico na dimensão é uma falta grave e deve ser diagnosticada o quanto antes no processo de carga. Por fim, o sexto problema, é identificado o erro para o tratamento de histórico para atributos com comportamento tipo 2 (armazenamento de histórico). Esse erro infringe a regra de atualização e armazenamento correto da data final, *flag* corrente e data inicial para os registros que não são mais correntes ou válidos atualmente.

Tabela 1: Grau de criticidade de erros contabilizados para cada programador.

	UC - Funcionário		
		Total de Grau Crítico	
	Problemas	Manual	Automático
Programador1	1; 4	91	0
Programador 2	2; 6	161	0
Programador 3	4	64	0
Programador 4	2; 5; 6	286	0
Programador 5	2	36	0
Programador 6	3; 6	152	0
Programador 7	4	64	0
Programador 8	2; 5; 6	286	0
Programador 9	3;4;6	216	0
Programador 10	5	125	0

A Tabela 1 mostra os resultados para o Caso de Uso de funcionários. Percebe-se que os programadores apresentaram um ou mais tipos de erros para implementarem os casos de testes de forma manual. A média de grau de criticidade de erros dos programadores foi 148,1. De forma automática através do FTUnit, a média foi 0 (zero), ou seja, não foram encontrados erros ao serem executados os procedimentos, para os casos de testes, criados por cada programador.

Pela constância da ausência de erros na execução automática dos testes e consequente não normalidade dos dados, já que as diferenças destes erros não se

distribuem normalmente, foi aplicado o Teste de Wilcoxon como Teste de Hipótese. Esse se caracteriza como não paramétrico, para amostras emparelhadas, levando em consideração que a amostra tem um comportamento contínuo e simétrico. O teste, além de comparar a diferença das amostras, também verifica a magnitude dessa diferença.

Com aplicação do Teste de Wilcoxon verificou-se que o Sig. de 0.005 (Tabela 2) é menor que o nível de significância de 0.05, ou seja, confirmou-se a evidência de diferença entre as médias de grau de criticidade de erros, para o UC01. Para metade dos procedimentos desenvolvidos de forma manual, não foram apresentados erros. Acredita-se que este resultado se deu pelo fato de ser um Caso de Uso simples, que apresentava poucas restrições, ficando mais perceptível para o programador detectar e implementar todas as especificações. Assim, confirmou-se a evidência de diferença entre as médias de grau de criticidade de erros.

Tabela 2: Teste Wilcoxon em relação aos erros para UC01 (Fonte: Ferramenta SPSS - IBM).

	ErrosUC01Automatico - ErrosUC01Manual
Z	-2,807
Asymp. Sig. (2-tailed)	**,005**

De forma geral, diante dos resultados, confirmam-se evidências de que os erros de execução de testes de forma manual e automática são significativamente diferentes, para o caso de uso apresentado. Logo, a hipótese (H_0) de que a execução de testes automática e manual tem a mesma eficácia foi rejeitada. Sendo assim, podemos acatar a hipótese alternativa de que a execução de testes de forma automática é mais eficaz que a manual.

CAPÍTULO 6

6 EXPERIMENTO II

***Abstract*—** This paper presents an approach to automate the selection and execution of previously identified test cases for loading procedures in Business Intelligence (BI) environments based on Data Warehouse (DW). To verify and validate the approach, a unit test *framework* was developed. The overall goal is achieve efficiency improvement. The specific aim is reduce test effort and, consequently, promote test activities in data warehousing process. A controlled experiment evaluation was carried out to investigate the adequacy of the proposed method against a generic *framework* for data warehouse procedures development. Constructed specifically for database application tests, DbUnit was the generic *framework* chosen for the experiment. The results of the experiment show that our approach clearly reduces test effort when compared with execution of test cases using a generic *framework.*

***Keywords*—** *Business Intelligence; Data Warehouse; Software Testing; Data Quality; Experimental Software Engineering; DbUnit.*

I. INTRODUCTION

Information represents a crucial factor for companies in improving processes and decision making. To assist the strategic areas of the organizations business intelligence (BI) environments are presented as sets of technologies that support the analysis of data and key performance indicators [1].

A central component of BI systems is a Data Warehouse (DW), a central repository of historical data. The idea behind this approach is to select, integrate and organize data from the operational systems and external sources, so they can be accessed more efficiently and represent a single view of enterprise data [1, 2, 3].

Despite the potential benefits of a DW, data quality issues prevent users from realizing the benefits of a business intelligence environment. Problems related to data quality can arise in any stage of the ETL (Extract, Transform and Load) process, especially in the loading phase. The main causes that contribute to poor data quality in data warehousing are identified in [4].

The lack of availability of automated unit testing facility in ETL tools is also appointed as cause for the poor data quality [4]. The low adoption of testing activities in DW environment is credited to the differences between the architecture of this environment and architectures of the generic software systems. These differences mean that the testing techniques used by the latter need to be adjusted for a DW environment [5, 6].

Tests of ETL procedures are considered the most critical and complex test phase in DW environment because it directly affects data quality [7]. ETL procedures, more precisely the loading routines, exhibit the same behavior as database applications. They operate on initial database state and generate a final consistent database state. So, a black-box approach, which combines the unit and application behavior of loading procedures, is proposed. In this approach, the concern is with the application interface, and not with the internal behavior and program structure [8,9,10]. This approach to ETL routines, in some environments, may be the only option, since the use of ETL tools in DW environment produces codes or packages whose internal structure is not known.

Previous studies was presented an experimentation with the proposal of using a unit testing *framework* (FTUnit) for loading routines in a BI environment based on Data Warehouse [23]. The motivation for adopting this approach meets the problems pointed out in [24], as the main causes for the poor quality of data in a DW environment. Through a set of metadata that defines the characteristics of the routines, the *framework* selects test cases to be applied, generates the initial states of the database, executes the routines, performs test cases, analyzes the final state of the database and generates a report with the errors encountered during the execution of each test case. With good results the use of the *framework* presents important contributions to increasing the productivity and quality in software engineering for loading routines of DWs.

This paper aims to show the results of an experiment to verify the best performance using the test *framework* (FTUnit) against a generic database application test *framework* (DbUnit). The performance results of FTUnit already presented in [23] shows the *framework* can accelerate and improve the quality of ETL process tests based on SQL. Now to compare with this study, the same use case was used for run the test cases at DbUnit in BI environment, once this *framework* had been constructed specifically for database application tests.

Thus, considering an industrial environment, this work aims address the following research question: "*in a load context for DW, test cases perform better using the test framework compared with a generic framework?*" To answer, our experimental evaluation analyzed a real context. The results showed numbers that indicate effort reduction using the teste *framework.*

The remainder of this paper is structured as follows. Section 2 presents the test *framework* and DbUnit. Section 3 describes the experiment definition and planning. Section 4 presents the operation of the experiment. Section 5 reports the results of the experiment. In Section 6, related works are presented. Finally, section 7 contains the conclusion and future works.

II. TESTING FRAMEWORK AND DBUNIT

A. Testing *Framework*

Unit testing *framework* (FTUnit) is used to perform unit tests in loading procedures of a DW environment. It has been developed in C#. It is available for download at <http://ftunit.wordpress.com/>. This tool resembles a *framework* for performing procedures on test cases in T-SQL (Transact-SQL) language. Therefore, it can be expanded to other SQL languages.

This *framework* will be used to perform tests under the black-box approach. The code and the internal structure of the routines will not be examined. The test cases previously implemented, will be selected according to the characteristics of the routine being tested. Each routine, in order to be covered by the *framework*, must have a set of metadata registered. This set of metadata was defined from the schema presented in [11].

B. DbUnit

DbUnit is a JUnit extension targeted at database-driven projects that puts the database into a known state between test runs. This can avoid the myriad of problems that occurs when one test case corrupts the database and causes subsequent tests to fail or exacerbate the damage [22].

Created to implement database operations tests in Java, DbUnit can work with large datasets when used in streaming mode and verify data matches an expected set of values. Using a XML based mechanism for loading test data, DbUnit can export existing test data into the XML format for subsequent use in automated tests. This method compares data, between database tables, flat files and queries [22].

III. EXPERIMENTAL DEFINITION AND PLANNING

Our work is presented here as an experimental process. It follows the guidelines by Wohlin et al. in [12]. In this section, we start introducing the experiment definition and planning. The following sections, will direct to the experiment execution and data analysis.

A. Goal definition

Our main goal is to evaluate the best performance using the test *framework* against a generic database application test *framework* in a Data Warehouse environment.

The experiment will target developers of ETL processes for BI environments with at least 2 years of experience in the market and one year of experience in ETL programming. The goal was formalized using the GQM model proposed by Basili and Weiss [13]:

- **Analyze** the use of a DW unit testing *framework*
- **With the purpose of** evaluate (against a generic database application test *framework*)
- **With respect to** the efficiency of the process of executing test cases
- **from the point of view of** developers and decision support managers
- **in the context of** programmers in a BI company.

B. Planning

1) Hypothesis Formulation

The research question for the experiment that needs to be answered is this: "*in a load context for DW, test cases perform better using the test framework compared with a generic framework?*"

To evaluate this question, it will be used a measure: Average time for Testing *Framework* and Generic *Framework*. Having the purpose and measures defined, it will be considered the hypothesis:

H_{0time}: the execution of test cases for the testing *framework* and the generic *framework* has the same efficiency. ($\mu_{Generic\mathit{Framework}Time} = \mu_{Tsting\mathit{Framework}Time}$).

H_{1time}: the execution of test cases for the testing *framework* is more efficient than running on generic *framework*. ($\mu_{Generic\mathit{Framework}Time} > \mu_{Tsting\mathit{Framework}Time}$).

Formally, the hypothesis we are trying to reject is H_{0time}. To ascertain which of the hypotheses is rejected, will be considered the dependent and independent variables that are in Figure 1.

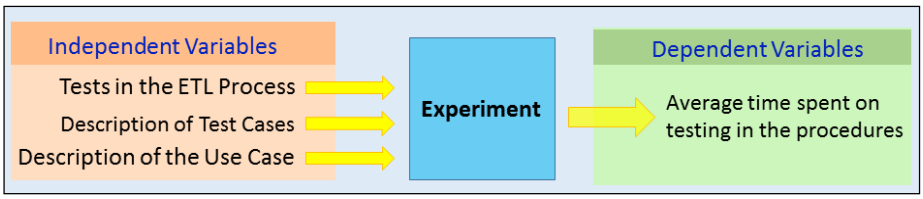

Figure 1. Dependent and Independent variables of the experiment.

a) Independent Variables

Next, the independent variables of the experiment are described.

Description of Test Cases Used in the Experiment: The loading routines for the DW environment are quite discussed in [1,2,14,15]. Alternative approaches to the loading of dimensions can be found in [16]. Algorithms for loading routines for the various types of dimensions can be found in [17]. Test Cases categories for ETL routines are pointed in [6,18]. This material, together with the extensive experience of the authors in DW projects in the public and private sectors, provided the basis for the elaboration of categories and test cases to be considered by the *framework*. The following categories are contemplated by the *framework*: a) Unit tests and relationship; b) Number of records between source and destination; c) Transformations between source and destination; d) Processing of incorrect or rejected data; e) Null values processing; f) Behavior type 1 and type 2 for dimensions attributes; g) Hybrid approaches for the treatment of historical dimensions.

TABLE I. FEATURES OF THE DIMENSION DBO.DIM_EMPLOYEE AND ITS ATTRIBUTES REGARDING THE HISTORICAL STORAGE.

Dimension Name:	dbo.dim_employee	
Treatment historical Type:	Type 1 and 2	
Attribute	Historical Type	Attribute Paper
id_employee		**Surrogate Key**
cod_registration		**Primary Key**
name	**1**	
cpf	**1**	
title	**2**	
job	**2**	
salary	**2**	
sector	**2**	
department	**2**	
dt_initial		**Initial Date**
dt_end		**End Date**
fl_current		**Current Flag**

Description of the Use Case Used in the Experiment: the characteristics of the use case chosen for the validation study were based on practical situations reported by the selected programmers.

For the use case of the experiment, the goal was to generate procedure to perform loads of Staging Area, from employee table to the employee dimension. At this time, the dimension has an historical storage, Type 2 for some attributes. The other attributes are Type 1.

Table I shows the characteristics of the employee dimension in DW environment. For the Type 2 treatment in dimensions, new attributes are used for the historical storage. They are: the start date, which represents the date on which the record was recorded; the end date which is the date when the record is no longer current; and finally, the attribute that identifies whether it is or not a current record. These are respectively shown in Table I, with the names of: dt_initial; dt_end and fl_current.

Tests in the ETL Process: The ETL tests used in this work, have two types of treatments for performing the experiment: 1) Generic *Framework*: test cases execution using DbUnit for loading data based on use case already presented earlier in a DW environment; 2) Testing *Framework*: execution of tests based on test case, in the SQL code for the same use case, using the proposed tool of this work, FTUnit.

DbUnit was select to this experiment first to have a *framework* proposal that is closest to the Testing *Framework*. Second, for convenience the experiment's programmers are already familiar with JUnit, so would be easier and efficient its use.

b) Dependent Variables

It were used a measure as a dependent variable: Average time, for testing *framework* and generic *framework*, measured using a stopwatch, considering the average time spent on testing in the procedures.

2) Participants Selection

The selection process of the participants will be done for convenience, making the type of sampling per share in which will be preserved the same characteristics of interest present in the population as a whole. The contributor to be chosen will be Qualycred (www.qualycred.com), a company that provides consulting in BI solutions for industry. This company will provide for the execution of the experiment, eight programmers with four years of experience in other areas and one year of experience working specifically with ETL for DW, in SGBD SQL SERVER.

3) Experiment Project

The experiment was projected in a paired context, in which a group will evaluate both approaches: Testing *Framework* and Generic *Framework* execution. For understanding the execution of the test to be done, ten test cases and one use case (seen in Independent Variables section) were elaborated, which will be presented in a well detailed way to the programmers.

The experiment will be separated into two groups of participants. Will be drawn 5 programmers to start the tests to the rules presented in the Employee Use Case, with the execution of Testing *Framework* and, shortly after, the execution of Generic *Framework*. The other participants in parallel, will make the tests made to rules presented in same use case, with the implementation of the Generic *Framework* and, shortly after, with the execution of the Testing *Framework*. Thus, the randomness will be enhanced, not prioritizing the manual or automated learning.

4) Instrumentation

The instrumentation process initially proceeded with the environment setup for the experiment and planning the data collect. It was conducted in a computer lab at Federal University of Sergipe - UFS. The Participants of the experiment had the same working conditions. The computers were adjusted to possess same settings. Listed below are the technology, the installed tools and artifacts used.

SQL Server 2008. It served as a basis for the storage of the identified metadata, and consequently, has been used to store the metadata of FTUnit, described in section 2.

Unit Testing Framework in Sql Code (FTUnit). The FTUnit was described section 2 of this paper. The version to be used in the experiment, due to the participants, runs Unit Testing Cases for charging procedures in SQL code, T-SQL language (Transact-SQL), involving behaviors of Types 1 and 2 for the treatment of historical dimensions.

DbUnit. Using XML files to run test cases in DbUnit, this framework has been prepared to run tests case in ETL procedures to load data to dimensions involving behaviors of type 1 and 2. At solution the programmers developed methods that could perform test cases for the employee use case, by means of an XML file with characteristics of employee auxiliary table. To compare the results after loading the data, it was compared with other XML file containing the expected characteristics of the employee dimension, thus confirm whether or not if the data were properly loaded for the dimension.

Environment Created and Produced Artifacts. Some of the tables that make up the DW environment used in the experiments were the following: a) auxiliary employee table; b) dimension of employee with attributes types 1 and 2. These are described below.

Figure 2 contains the representation of a load data from TB_Aux_Employee (auxiliary table of employees) to the DIM_Employee (dimension of employees) that represents a dimension of types 1 and 2. To this dimension, the attributes that match the type 1 are: name and CPF. The attributes of type 2 are: title, job, salary, sector and department.

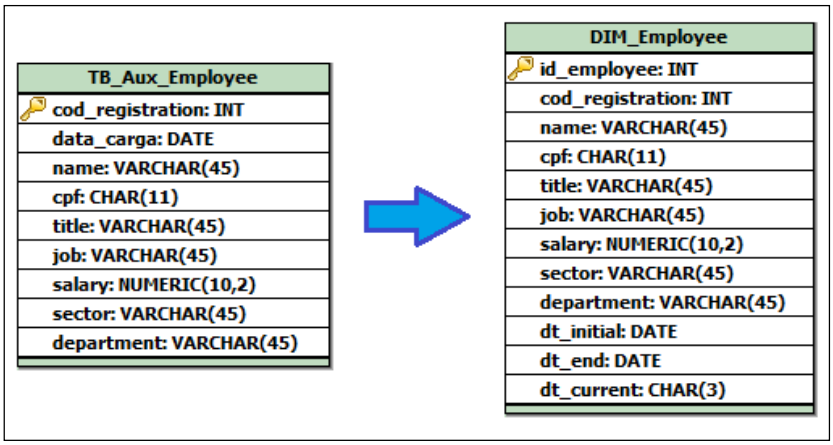

Figure 2. Charge for a dimension of employees with behaviors of types 1 and 2.

IV. EXPERIMENT OPERATION

A. The preparation

The following are listed the preparation steps for the execution of the experiment.

1) *DW environment Creation:* in this phase was defined and created the DW environment with the dimensional schemes and staging area. These artifacts served as the basis for the entire experiment.
2) *Definition of Test Cases for loading routines:* were defined test cases to be followed by the developers of the experiment. These test cases were applied to a loading routine previously created.
3) *Review of basic concepts of loading routines for the programmers:* a review of the loading routines, for DW environments with the selected developers, was performed.
4) *Training in Testing Framework (FTUnit):* a training with the programmers was realized to become familiar with the tool.
5) *Training in Generic Framework (DbUnit):* a training with developers was conducted for tool learning in the context of charges for DW environments.

In short, all computers were prepared with the same settings, so programmers were on the same working conditions. Moreover, it was presented to each programmer, a printed document containing a detailed description of Use Case and test cases that would be used by them, in case of any doubts.

B. Execution

At the end of the previous steps, the experiment was initiated, it occurred according to the plan described in section 3.

The evaluation of Testing *Framework* at the end of the experiment, made by the professionals, was positive, since they have commented that the use of this tool have contributed to the reduction of time in the test procedures.

1) Data Collection

Average time for Testing *Framework* and Generic *Framework.*

It was calculated the time spent by each developer for Testing *Framework* and Generic *Framework* tests, of all test cases for the Employee Use Case, taking into account the time for testing and all necessary settings in FTUnit. Under supervision, each programmer reported the completion and was recorded the time on a timer, used for this purpose. The result of these collected data will be presented in section 5 of this paper.

C. Data Validation

In order to perform the experiment, one factor was considered, Test of the ETL Process, and two treatments, execution of tests case using the FTUnit tool and DbUnit tool. Facing this context, the average of testing time was computed.

As an aid to analysis, interpretation and validation, we used two types of statistical tests, Shapiro-Wilk Test and the T Test. Shapiro-Wilk test was used to verify normality of the samples. The T test was used to compare the average of the two paired samples [12]. All statistical tests were performed using the SPSS tool [19].

V. RESULTS

A. Analysis and Data Interpretation

To answer the question of research, the following dependent variable was analyzed: The time to the testing process of each procedure.

1) Time spent in the testing process.

Table II displays the results related to the testing time by each participant for the Employee Use Case. The results show that the average time of the developers for Testing *Framework* was 23.13 minutes, and 166.38 minutes for the Generic one.

These results suggest that the Testing *Framework* execution have, on average, shorter testing time, as compared with the same test procedure performed in Generic *Framework* by programmers with experience in the area. Thus, from this preliminary analysis of the data, it is assumed that the answer to the Research Question would be "yes". The execution of test cases using the FTunit can increase the productivity of developers during the testing process in a DW compared with DbUnit, since this tool obtained a difference of approximately 143.25 minutes. But is not possible to make such a claim without sufficiently conclusive statistical evidence.

Thus, first, we established an apriority significance level of 0.05. The Shapiro-Wilk test ensured that the sample was normally distributed. As seen in Table III, we found p-values of 0.672 and 0.523 for the use of testing *framework* and generic *framework*, respectively. As the p-value is the lowest possible significance with which it is possible to reject the null hypothesis, and they are larger than 0.05, we cannot reject the hypothesis that the data is normally distributed.

Finally, as the samples are not independent, the hypothesis test applied in this context was the T-Test, characterized as parametric for paired samples, which only requires normality of the samples. In Table IV, we obtained the p-value of 0.000. This means the p-value found is less than 0.0001, so we have more than 99% certainty for the valued context. Thus, it was confirmed the evidence of a difference between the averages of 143.25. As the significance test is lower than 0.05, it is possible to reject the null hypothesis. Consequently, we cannot reject the alternative hypothesis that the execution of test cases for the testing *framework* is more efficient than running on generic *framework*.

TABLE II. AVERAGE TIME (IN MINUTES) TO EXECUTE THE TESTS OF THE USE CASES.

	Employee Use Case	
	Testing *Framework* (minutes)	**Generic *Framework* (minutes)**
Programmer 1	23	141
Programmer 2	27	154

Programmer 3	19	82
Programmer 4	20	161
Programmer 5	28	209
Programmer 6	21	182
Programmer 7	22	186
Programmer 8	25	216

TABLE III. SHAPIRO-WILK TEST. (SOURCE: SPSS TOOL – IBM [19]).

	Statistic	**df**	**Sig.**
FTUnit	,946	8	,672
DbUnit	,931	8	,523

TABLE IV. T-TEST VERSUS TIME OF THE TESTS. (SOURCE: SPSS TOOL – IBM [19]).

		Paired Differences					t	df	Sig(2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference				
					Lower	Upper			
Par 1	DbUnit-FTUnit	143,25	41,053	14,514	108,929	177,571	9,869	7	,00

B. Threats to validity

In spite of having achieved statistical significance in the study, the following threats to the validity must be considered.

Threats to internal validity: the limited availability of programmers to make new use cases can be considered a threat to validity for having implemented a simple use case. Also although participants have been trained to use both *framework*s, they do not use it daily. This lack of constant contact with it may have affected the results, which could be even better, pro-tool. *Framework*s training were conducted at the beginning of the experiment, considering a phenomenon studied by psychology called *Demand Characterization* - which considers that an experimental artifact may have an interpretation of the purpose of the experiment by the participants. This can lead to

change of unconscious behavior, to adapt to this interpretation [20]. According to this concept, this training could be harmed the progress of the experiment, but to mitigate this factor, can be said that had been used at least two different approaches*: The More The Merrier* and *Unobtrusive Manipulations and Measures* [20]. Respectively, the first, to avoid bias with a single experimenter, the experiment had another researcher to conduct the experiment and an instructor for the tool, not involved with the research. The second guided us not to say which factors and metrics would be assessed, so that the participants had no clues about the research hypothesis.

Threats to external validity: The low number of participants can be a threat, since it can negatively influence the results of the experiment. This threat was mitigated with the convenience of the selection of programmers skilled in the ETL area for BI environment.

Threats to the construction validity: The Specifications for the use case and test cases may not have been very clear to the understanding of some programmers. This threat was mitigated with the prior reading and analysis of the understanding, made by four ETL developers.

VI. RELATED WORK

Through literature reviews, with systematic approaches, were not found strongly related work for automated unit tests in ETL tools. Consequently, the absence of ETL tools with these characteristics may contribute to a lower integrity and a lower quality of data, essential in large banks of decision support data.

Some moderately related works also seek solutions for the automatic execution of Test Cases in DW environments. In [6] it is presented a directed models approach for automatic generation and execution of test cases based in formal models of systems. The formal model adopted is based on the UML language. The approach also depends on creating an extension of UML language that can capture the transformations used in a *Data Warehousing* process.

Once the Testing *Framework* generates test cases based on characteristics of the loads procedures being implemented, it can be extended and used to test load routines

created for any ETL tools. So far was not found in literature any similar approach, so we could make a comparison.

It's possible find in [21] an evaluation of unit testing tools suitable for data warehouse testing. The following open source tools were select based on tools that could perform test cases in a BI environment: AnyDbTest, BI.Quality, DbFit, DbUnit, NDbUnit, SQLUnit, TSQLUnit, and utPLSQL. The more similar tool to the proposed work is the DBUnit *framework* [22]. However, this one represents a generic *framework* for database applications and has no particularity regarding to loading routines for a Data Warehouse environment.

VII. CONCLUSIONS

Business Intelligence requires valid, consistent, and complete organizational data. These quality items represent constant concerns for companies in the process of use of decision support systems.

In this paper, we presented the proposal of using a Unit Testing *Framework* and a Generic *Framework* for loading routines in a BI environment based on Data Warehouse. The experimentation's results show the use of FTUnit was more efficient than the use of DbUnit. The motivation for adopting this approach meets the problems pointed in [6,7,8] is the need to adopt different strategies, considering the differences between traditional environments and DW environments, which can contribute to the adoption of testing processes.

In this context, this work presents important contributions to increasing the productivity and quality in software engineering for loading routines of DWs, and encourages experimentation in an industrial environment. The Testing *Framework* encapsulates a method to accelerate and improve the quality of ETL process tests based on SQL. It is noteworthy that the safe and efficient execution of procedures in SQL directly in the database is an option considered by much of the industry, requiring tools to support tests in this type of approach in software engineering.

Although this study did not show satisfactory results in the experiment for the use of DbUnit, a new approach has been set to perform test cases in DW environment. Therefore, it becomes something new in the area, since were not found any work

containing the applicable implementing testing procedures in a BI environment for a generic *framework* as DbUnit.

The proposed *framework* (FTUnit) presents test cases previously defined which cover the main categories of tests applied to loading routines. Through a set of metadata that defines the characteristics of the routines, the *framework* selects test cases to be applied, generates the initial states of the database, executes the routines, performs test cases, analyzes the final state of the database and generates a report with the errors encountered during the execution of each test case.

By virtue of what we have seen above and the *framework* innovation, the presentation of this experiment will support the adoption of the same or the creation of a similar approach for companies that use this type of strategy.

As future work, It aims to extend the approach to various SQL languages, as the experiments carried out so far have been only for the T-SQL.

REFERENCES

[1] Colaço Jr., M.: Projetando sistemas de apoio à decisão baseados em Data Warehouse. 1st ed., Rio de Janeiro: Axcel Books (2004)

[2] Kimball, R., Ross, R. M. and Thomthwaite, W.: The Data Warehouse lifecycle toolkit. 2nd. ed., Indianapolis, Indiana: Wiley Publishing Inc (2008)

[3] Inmon, W. H.: Building the Data Warehouse. 4th ed., Indianapolis, Indiana: Wiley Publishing Inc (2005)

[4] Ranjit S. and Kawaljeet, S.: A Descriptive Classification of Causes of Data Quality Problems in Data Warehousing. 7 v. IJCSI International Journal Of Computer Science Issues (2010)

[5] Deshpande, K.: Model Based Testing of Data Warehouse. IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 2, No 3 (2013)

[6] Elgamal, N., Elbastawissy, A. and Galol-edeen, G.: Data Warehouse Testing. EDBT/ICDT ’13, Genoa, Italy (2013)

[7] Golfarelli, M. and Rizzi, S. A.: Comprehensive Approach to Data Warehouse Testing. ACM 12th International Workshop on Data Warehousing and OLAP (DOLAP ’09), Hong Kong, China (2009)

[8] Myers, G. J., Badgett, T. and Sandler, C.: The Art Of Software Testing. 3rd ed., New Jersey: Wiley (2012)

[9] Pressman, R. S.: Engenharia de software: Uma abordagem profissional. 7th ed., São Paulo: AMGH Editora Ltda (2011)

[10] Sommerville, I.: Engenharia de Software. 9th ed., São Paulo: Pearson (2011)

[11] Costa, J. K. G.; Santos, I. P. O.; Nascimento, A. V. R. P.; Colaço Jr., M.;. Experimentação na Indústria para Aumento da Efetividade da Construção de Procedimentos ETL em um Ambiente de Business Intelligence. SBSI 2015, May 26–29, Goiânia, Goiás, Brazil (2015)

[12] Wohlin, C., et al.: Experimentation in Software Engineering: An introduction. USA: Kluwer Academic Publishers (2000)

[13] Basili, V. and Weiss, D.: A Methodology for Collecting Valid Software Engineering Data. In: IEEE Transactions On Software Engineering, v.10 (3): 728-738, November (1984)

[14] Kimball, R. and Ross, M.: The Data Warehouse toolkit: The complete Guide to Dimensional Modeling. 2nd ed., John Wiley and Sons, Inc (2002)

[15] Kimball, R.: The Data Warehouse ETL Toolkit. 1st ed., Wiley India (P) Ltd (2004)

[16] Santos, V. and Belo, O.: No Need to Type Slowly Changing Dimensions. IADIS International Conference Information Systems (2011)

[17] Santos, I. P. O.; Costa, J. K. G.; Nascimento, A. V. R. P.; Colaço Jr., M. Desevolvimento e Avaliação de uma Ferramenta de Geração Automática de Código para Ambientes de Apoio à Decisão. In: XII WTICG, XII ERBASE (2012)

[18] Cooper, R. and Arbuckle, S.: How to thoroughly test a Data Warehouse. Proceedings of STAREAST, Orlando (2002)

[19] SPSS, IBM Software, http://goo.gl/eXfcT3

[20] Orne, M. T.: Sobre a psicologia social da experiência psicológica: Com referência particular para exigir características e suas implicações. (1962)

[21] Krawatzeck, R.; Tetzner, A. and Dinter, B. An Evaluation Of Open Source Unit Testing Tools Suitable For Data Warehouse Testing. The 19th Pacific Asia Conference on Information Systems (PACIS), 2015.

[22] DbUnit, http://DbUnit.sourceforge.net/

[23] Santos, I. P. O.; Costa, J. K. G.; Nascimento, A. V. R. P.; Colaço Jr., M.; Pereira, W. C. Experimentation in the Industry for Automation of Unit Testing in a Business Intelligence Environment. The Twenty-Eighth International Conference on Software Engineering and Knowledge Engineering (SEKE), San Francisco (2016)

CAPÍTULO 7

7 CONCLUSÕES

7.1 Conclusões

Identificar e tentar solucionar os problemas de qualidade de dados em um ambiente de *Data Warehouse* representa um dos principais obstáculos enfrentados pelas grandes empresas no processo de utilização de Sistemas de Apoio à Decisão. Dentre os vários fatores que contribuem para a má qualidade dos dados, estão os testes feitos de forma manual de rotinas de carga de dados.

Baseado nesse contexto, foram levantadas as seguintes questões de pesquisa neste trabalho:

1. A execução de testes automáticos pode aumentar a produtividade dos programadores durante o processo de teste em um DW?
2. A execução de testes automáticos pode reduzir ou eliminar erros nas rotinas de carga em um DW?
3. Em um contexto de carga para DW, os casos de testes apresentam melhor desempenho utilizando o *framework* de testes comparado com um *framework* genérico?

Após delimitar as perguntas anteriores, levantamos a hipótese de que os testes feitos com o apoio de *Framework* de Testes podem contribuir para melhoria da qualidade através do impacto em variáveis como produtividade e erros de codificação. Além disso, verificou se o *Framework* de Testes possui um melhor desempenho comparado com um *framework* de testes genérico (DbUnit).

Para acatar ou refutar as hipóteses apresentadas, foram planejados e executados dois experimentos controlados e executados na indústria. O primeiro avaliou a utilização de casos de testes para as rotinas de carga, comparando a efetividade do *framework* com uma abordagem manual. O segundo efetuou a comparação com um *framework* genérico similar existente no mercado.

O primeiro experimento apresentou fortes indícios de que as variáveis “Tempo Gasto no Processo de Teste” e “Número de Erros de Codificação” possuem relação com o tipo de tratamento empregado (questões de pesquisa 1 e 2). O segundo experimento apresentou bons resultados quanto à eficiência na utilização e execução dos Casos de Testes, utilizando o FTUnit e efetuando uma comparação com o DbUnit (questão de pesquisa 3). Desse modo, existem fortes evidências que o *Framework* de Teste de unidade para rotinas de carga pode contribuir para o aumento da produtividade e redução dos erros de codificação em ambientes de suporte à decisão.

O *framework* proposto apresenta casos de testes previamente definidos que cobrem as principais categorias de testes aplicados às rotinas de carga. Em virtude disso e da inovação do *framework*, esta dissertação embasará a adoção do mesmo ou a criação de uma abordagem similar por empresas que utilizam este tipo de estratégia.

7.1.1 Contribuições

As contribuições obtidas são:

- Abordagem para execução de testes de *software* em ambientes BI baseados em DW;
- Definição de conjunto de Casos de Testes para rotinas de carga de dados em ambientes de BI, os quais podem ser reaproveitados pelos profissionais e pesquisadores das áreas de testes e BI;
- *Framework* de Testes para atender à execução de testes de unidade num ambiente de BI;
- Documentação de como usar o DBUnit para execução de testes unitários em ambientes de BI baseados em DW; e,
- Experimentos que evidenciam benefícios dos testes automáticos em ambientes de BI.

7.2 Trabalhos Futuros

O presente projeto pode servir de base para trabalhos futuros. Primeiro, a geração de casos de testes para realização de testes em Tabelas de Fato, Hierarquias e Agregados é uma extensão importante que pode confirmar os benefícios da utilização de

um *Framework* de teste de unidade para procedimentos de cargas em um ambiente de *Data Analytics*. Ato contínuo, pretende-se estender a abordagem para diversos idiomas SQL, já que os experimentos realizados até o momento foram somente para o T-SQL.

REFERÊNCIAS

ABRANTES, J. **Gestão da Qualidade**. 1ª. ed. Rio de Janeiro: Interciência, 2009.

BASTOS, A.; RIOS, E.; CRISTALLI, R.; MOREIRA, T. **Base de Conhecimento em Teste de Software**. 3 ed. São Paulo: Editora Martins Fontes, 2012.

COLAÇO JÚNIOR, M. **Projetando sistemas de apoio à decisão baseados em data warehouse**. Rio de Janeiro: Axcel Books, 2004.

COOPER, R.; ARBUCKLE, S. **How to thoroughly test a data warehouse**. Proceedings of STAREAST. Orlando, 2002.

CORDEIRO, A. G.; FREITAS, A. L. P. **Priorização de requisitos e avaliação da qualidade de software segundo a percepção dos usuários**. [Editorial]. Ciência da Informação, Brasília, v. 40, n. 2, p. 160-179, maio/ago., 2011.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. **Introdução Ao Teste De Software.** 1 ed. São Paulo: Elsevier, 2007.

DESHPANDE K. **Model Based Testing of Data Warehouse**. IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 2, No 3, March 2013.

ELGAMAL, N; ELBASTAWISSY A; GALOL-EDEEN G. **Data Warehouse Testing**. EDBT/ICDT ’13, Genoa, Italy, March 2013.

GOLFARELLI, M; RIZZI S. **A Comprehensive Approach to Data Warehouse Testing**. ACM 12th International Workshop on Data Warehousing and OLAP (DOLAP ’09), Hong Kong, China, 2009.

GONÇALVES, L. F. V. **A Redução De Problemas De Utilização Do Método Ciclo Pdca : Um Estudo De Caso**. VII Congresso Nacional de Escelência em Gestão, p. 18, 2011.

IDRIS, N.; AHMAD, K. Managing Data Source quality for data warehouse in manufacturing services. **Proceedings of the 2011 International Conference on Electrical Engineering and Informatics**, n. July, p. 1–6, 2011.

IEEE. INTERNATIONAL STANDARD ISO / IEC / IEEE. v. 2010, p. 418, 2010.

INMON, W. H. **Building the data warehouse, Fourth Edition.** 4. ed. Indianapolis, Indiana: Wiley Publishing Inc., 2005.

JURISTO, N.; MORENO, A. M. **Basics of Software Engineering Experimentation**. Analysis, v. 5/6, p. 420, 2001.

KERLINGER, F. N.; LEE, H. B. **Foundations of Behavioral Research**. 2ª. ed. [s.l.] Cengage Learning, 1973.

KIMBALL, R. **The Data Warehouse ETL Toolkit.**1 ed. Wiley India (P) Ltd., 2004.

KIMBALL, R; ROSS, M. **The data warehouse toolkit: The complete Guide to Dimensional Modeling.** 2. ed. John Wiley and Sons, Inc., 2002.

KIMBALL, R; ROSS, M. THORNTHWAITE, W. **The data warehouse lifecycle toolkit**. 2 ed. Indianapolis, Indiana: Wiley Publishing Inc., 2008.

KOSCIANSKI, A.; SOARES, M. dos Santos. **Qualidade de software: aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software**. 2ª Edição. São Paulo: Novatec, 2007.

KRAWATZECK, R. **Pilotstudie: Praktiken und Herausforderungen beim Testen von BI-Systemen.** BI-Spektrum, 8(3), 10–14, 2013.

KRAWATZECK, R.; TETZNER, A. and DINTER, B. **An Evaluation Of Open Source Unit Testing Tools Suitable For Data Warehouse Testing.** The 19th Pacific Asia Conference on Information Systems (PACIS), 2015.

MARSHALL, I. J. et al. **Gestão da qualidade**. 10. ed. Rio de Janeiro: FGV, 2011.

MILLER, E., **Automated Tools for Software Engineering**, IEEE Computer Society Press, p. 169, 1979.

MYERS, G. J.; BADGETT, T.; SANDLER, C. **The ArT Of *Software* Testing**. 3. ed. New Jersey: Wiley, 2012.

PEZZÈ, M.; YOUNG, M. **Teste e Análise de Software: processo, princípios e técnicas**. Porto Alegre: Bookman, 2008.

PRESSMAN, R. S. **Engenharia de *software*: Uma abordagem profissional**. 7 ed. São Paulo: AMGH Editora Ltda, 2011.

RANJIT, S.; KAWALJEET , S. **A Descriptive Classification of Causes of Data Quality Problems in Data Warehousing**. IJCSI International Journal Of Computer Science Issues, 2010.

SANTOS, I. P. O.; COSTA, J. K. G.; NASCIMENTO, A. V. R. P.; COLAÇO JÚNIOR, M. **Desevolvimento e Avaliação de uma Ferramenta de Geração Automática de Código para Ambientes de Apoio à Decisão**. In: XII WTICG, Workshop de Trabalhos de Iniciação Científica e de Graduação Bahia-Alagoas-Sergipe, 2012, Juazeiro. XII Escola Regional de Computação Bahia Alagoas Sergipe - ERBASE 2012a.

SANTOS, I. P. O.; NASCIMENTO, A. V. R. P. **Desevolvimento e Avaliação de uma Ferramenta de Geração Automática de Código para Ambientes de Apoio à Decisão**. Relatório Final de Pesquisa do Programa Especial de Inclusão em Iniciação Científica (PIIC/UFS). Julho de 2012b.

SANTOS, V; BELO, O. **No Need to Type Slowly Changing Dimensions.** IADIS International Conference Information Systems, 2011.

SCHUTTE, S., ARIYACHANDRA, T. and FROLICK, M. **Test-Driven Development of Data Warehouses**. International Journal of Business Intelligence Research, 2(3), 64–73, 2011.

SEVERINO, A. J. **Metodologia do trabalho científico**. 23. ed. [s.l.] Editora, Cortez, 2008.

SINGH, R.; SINGH, K.**A Descriptive Classification of Causes of Data Quality Problems in Data Warehouse.** IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 3, No 2, May 2010.

SOMMERVILLE, I. **Engenharia de *Software***. 9 ed. São Paulo: Pearson, 2011.

TRISTÃO, R. G. C. **A importância das ações corretivas e ações preventivas nos sistemas de gestão da qualidade - um estudo em empresas certificadas ISO 9001 no estado do rio de janeiro**, 2011.